МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Савченко А.С.

«___»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТР"

Тема: «Web-додаток планування персональних завдань»

Виконавець: Костенко Станіслав Ігорович

Керівник: д.т.н., доц. Райчев Ігор Едуардович

Нормоконтролер з ЄСКД (ЄСПД): Райчев І.Е.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕСИТЕТ

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Спеціальність 122 “Комп’ютерні науки та інформаційні технології”

Спеціалізація «Інформаційні управляючі системи та технології (за галузями)»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Савченко А.С.

«___» _____2019р.

З А В Д А Н Н Я

на виконання дипломної роботи студента

Костенко Станіслав Ігорович

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи): «Web-додаток планування персональних завдань» затверджена наказом ректора №2175/ст. від 25.09.2019р.
2. Термін виконання проекту (роботи): з 26.09.2019р. по 03.02.2020р.
3. Вихідні данні до проекту (роботи): веб-додаток планування персональних завдань.
4. Зміст пояснювальної записки (перелік усіх її розділів): огляд основних принципів та технологій для побудови веб-додатків, аналіз архітектурних шаблонів та проектування майбутнього веб-додатку, реалізація веб-додатку.
5. Перелік обов’язкового графічного матеріалу: рисунки, діаграми.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1.	Аналіз літератури та джерел за темою дипломного проекту.	26.09.19р.– 20.10.19	
2.	Розробка та затвердження плану дипломного проекту.	21.10.19– 22.10.19	
3.	Проведення консультації з науковим керівником щодо створення першого розділу.	23.10.19 – 27.10.19	
4.	Розробка розділу 1:	30.10.19 – 22.11.19	
5.	Розробка розділу 2	23.11.19 – 08.12.19	
6.	Розробка розділу 3:	09.12.19 – 22.12.19	
7.	Висновки та оформлення пояснювальної записки дипломного проекту.	25.12.19 – 29.12.19	
8.	Підписання необхідних документів у встановленому порядку.	15.01.2020-27.01.2020	
9.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту.	27.01.2020 – 03.02.2020	

7. Дата видачі завдання: 25.09.2019р.

Керівник дипломного проекту _____ Райчев І.Е.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Костенко С.І.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Web-додаток планування персональних завдань» викладена на с., містить рис., табл., літературних джерел.

Ключові слова: ВЕБ-ТЕХНОЛОГІЇ, ВЕБ-СТОРІНКА, БРАУЗЕР, РОЗРОБКА ДОДАТКУ, ВЕБ-ДОДАТОК, АРХІТЕКТУРА, ДОДАТОК.

***Об'єкт дослідження*:** процес проектування веб-додатку.

Предмет дослідження: веб-додатку для планування персональних завдань.

Мета роботи: розробка функціонального веб-додатку

Методи дослідження: аналіз літератури та моделювання необхідних процесів проектування

Отримані результати: реалізовано локальний одно сторінковий веб-додаток для планування персональних завдань.

Результати дипломної роботи використовуються на підприємстві та в повсякденному житті.

ОГЛАВЛЕНИЕ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CSS - Cascading Style Sheets

JS - Java Script

URL – Uniform Resource Located

HTML - Hypertext Markup Language

XML - Xtensible Markup Language

HTTP - HyperText Transfer Protocol

API - Application Programming Interface

DOM - Document Object Model

SPA - Single Page Application

ВСТУП

Даний дипломний проект вирішує проблему комплексного підходу до проектування веб-додатку планування персональних завдань, який необхідний для постійного відслідковування часових рамок завдань. Тобто додаючи необхідне завдання , користувач має можливість встановити час та дату для визначення строку його виконання. Цими діями користувач полегшує собі життя, адже вже не потрібно постійно пам’ятати про строк виконання окремих задач. В даному веб-додатку, не обов’язково записувати чіткі задачі та їх строк. Це характеризує можливість створювати будь-які користувацькі списки.

Спискам справ характерні як побутові ситуації так і робочі, навіть для роботи великих компаній. Адже працівники більшості компаній обов’язково планують свій день на нараді, на початку дня.

В даний час широко поширені мобільні додатки, такі як органайзер, заснований на завданнях, що є складовою розпорядок дня. Тому в мережі наявна доволі велика кількість таких додатків, які мають розповсюджену назву TODO-list. Але в більшості випадків ці додатки створюють на початку ознайомлення з програмуванням або якоїсь нової технології.

Використовуючи створений додаток локально, тобто без завантаження на веб-сервер, ми отримуємо від нього 2 корисні властивості:

1. Це організація розпорядку дня, тобто відображення структури дня та запис усіх завдання, щоб нічого не залишилося без уваги.
2. Це наявність робочого проекту який ми має можливість постійно вносити зміни.

Тим самим пробуючи нові методи та засоби для реалізації певних модулів, або реалізовуючи нові підходи чи технології для створення програмні засоби. Тобто, маючи такий додаток в своєму арсеналі, ми маємо можливість постійно підтримувати його актуальним у використанні веб-

технологій розробки, завдяки реалізації нового функціоналу або масштабування самого проекту.

Результатом дипломного проекту – реалізований та описаний повний цикл створення веб-додатку для планування персональних завдань, а також задокументовано основні та важливі процеси розробки.

РОЗДІЛ 1 ОГЛЯД ОСНОВНИХ ПРИНЦИПІВ ТА ТЕХНОЛОГІЙ ДЛЯ ПОБУДОВИ ВЕБ-ДОДАТКІВ

1.1 Поняття веб-додатків

Веб-додаток – це будь-який веб-сайт з елементами інтерактиву. Веб-додаток являє собою веб-сайт, на якому розміщені сторінки з частково або повністю несформованим вмістом. Остаточний вміст формується тільки після того, як відвідувач сайту запросить сторінку з веб-серверу. У зв’язку з тим що остаточний вміст сторінки залежить від запиту, створеного на основі дій користувача, така сторінка називається динамічною. Спектр використання веб-додатків досить широкий. Веб-дизайн та додатки передбачають стандарти для створення та надання веб-сторінок, включаючи HTML, CSS, SVG, API пристроїв та інші технології для веб-додатків.

Використання веб-додатка замість типової веб-сторінки приносить користь для користувача.

1. Веб-додатки дозволяють відвідувачам швидко і легко знаходити необхідну інформацію на веб-сайтах з великим об'ємом інформації.
2. Веб-додатки дозволяють збирати, зберігати і аналізувати дані, отримані від відвідувачів сайту.
3. Веб-додаток може використовуватися для оновлення веб-сайтів з динамічно створюваним вмістом.

Перший вид додатків дозволяє здійснювати пошук у вмісті, упорядковувати вміст і переміщатися по ньому зручним для відвідувачів способом.

Другий тип додатків доволі довго використовував метод при якому дані, введені в HTML-форми, відсилалися для обробки CGI-додатків або спеціально

призначеним працівникам у вигляді повідомлень електронної пошти. Веб-додаток дозволяє зберігати дані безпосередньо в базі даних, а також отримувати дані і формувати звіти на основі отриманих даних для аналізу.

Наприклад можна привести інтерактивні сторінки банків, сторінки для контролю товарних запасів, соціологічні дослідження та опитування, а також форми для зворотного зв'язку з користувачами.

Результатом виконання дипломного проекту повинен стати третій тип веб-додатків. Даний тип характерний тим, що інформація яка виводиться на екран, створюється динамічно.

1.1.1 Принципи роботи веб-додатків

Для початку необхідно визначити різницю між веб-сайтом та веб-додатком. Веб-сайт це в першу чергу щось інформаційне і статичне: візитка компанії, сайт рецептів, міський портал або *Wiki*. Набір підготовлених заздалегідь HTML-файлів, які лежать на віддаленому сервері і віддаються браузеру за запитом.

Веб-сайт показує статичні або динамічні дані, які переважно надсилаються з сервера лише користувачеві, тоді як веб-додаток обслуговує динамічні дані з повною двосторонньою взаємодією.

Веб-сайт показує по суті ті ж дані. Деякі з них можуть бути динамічними, але це, як правило, справа в одну сторону - ви лише споживач.

Веб-додаток - двосторонній; ви бачите дані, які не тільки динамічні, але часто також специфічні для вас. Ви можете працювати з цими даними через веб-додаток, щоб публікувати новий вміст або надсилати змістовні запити назад на сервер або через сервер третім особам (включаючи інших користувачів).

Наприклад;

- додаток для продажу акцій / акцій з даними про ціни та рахунок у режимі реального часу, що дозволяє здійснювати операції в режимі реального часу.
- додаток для редагування фотографій.
- генератор списку бажань на весілля, який можна поділитися з вашими гостями
- веб-гра зі стійким світом
- Інтерфейс публікації веб-сайту Wordpress (де ви пишете дописи в блозі та загалом керуєте сайтом) - це веб-додаток, але сам блог це не так .
- Youtube після входу (щоб ви могли розміщувати коментарі чи відео).

Що таке веб-сайт, загально прийнято, але те, що також не відноситься до веб-додатком, є трохи сірою зоною; Наприклад Google Maps - це не веб-додаток (це динамічна веб-сторінка з пошуком і фільтруванням), тоді як пошта Google - справжня веб-програма, і це велика частина соціальних мереж.

Для чіткого розуміння принципу роботу веб-додатків, розглянемо як працює WEB на даний момент. Комп'ютери, підключені до мережі називаються клієнтами та серверами. Спрощена схема того, як вони взаємодіють зображено на Рис.1:

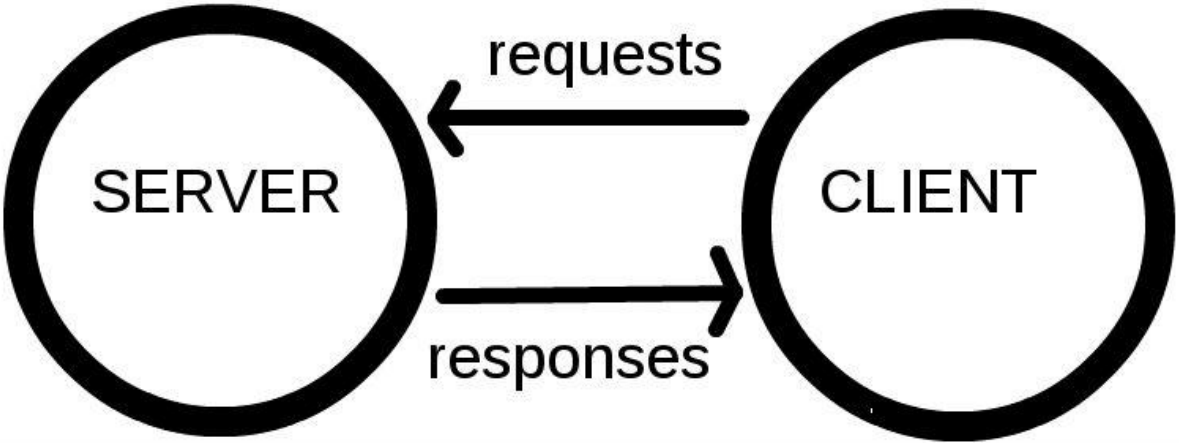


Рис. 1 Архітектура "Клієнт-сервер"

Клієнти є звичайними користувачами, підключеними до Інтернету за допомогою пристроїв (наприклад, комп'ютер підключений до Wi-Fi, або ваш телефон підключений до мобільної мережі) і програмного забезпечення, доступного на цих пристроях (як правило, браузер, наприклад, Firefox або Chrome).

Сервери - це комп'ютери, які зберігають веб-сторінки, сайти або додатки. Коли клієнтський пристрій намагається отримати доступ до веб-сторінці, копія сторінки завантажується з сервера на клієнтський комп'ютер для відображення в браузері користувача.

Але клієнт та сервер, не розкривають всю суть, адже ще використовується багато інших компонентів. Для більш чіткого розуміння, представимо, що WEB – це дорога. Одна сторона дороги є клієнтом, який являє собою ваш будинок. Інша сторона дороги є сервером, який являє собою магазин. І ви хочете щось купити в ньому. Тому необхідно приділити увагу:

- Ваше Інтернет-підключення: Дозволяє відправляти і приймати дані по мережі. Подібне до вулиці між будинком і магазином.
- TCP / IP: Протокол Управління Передачею і Інтернет Протокол є комунікаційними протоколами, які визначають, яким чином дані повинні передаватися по мережі. Вони як транспортні засоби, які

дозволяють зробити замовлення, піти в магазин і купити ваші товари. У нашому прикладі, це як автомобіль або велосипед (або власні ноги).

- DNS: Система доменних Імен схожі на адресну книгу для веб-сайтів. Коли ви вводите веб-адресу в своєму браузері, браузер звертається до DNS, щоб знайти реальну адресу веб-сайту, перш ніж він зможе його отримати. Браузеру необхідно з'ясувати, на якому сервері живе сайт, тому він може відправляти HTTP-повідомлення в потрібне місце (див. Нижче). Це схоже на пошук адреси магазину, щоб ви могли потрапити в нього.
- HTTP: Протокол Передачі Гіпертексту - це протокол, який визначає мову для клієнтів і серверів, щоб спілкуватися один з одним. Він, як мова, яку ви використовуєте, щоб замовити ваш товар.
- Файли компонентів: сайт складається з декількох різних файлів, які подібні до різних відділах з товарами в магазині. Ці файли бувають двох основних типів:
    - Файли коду: сайти побудовані переважно на HTML, CSS і JavaScript.
    - Матеріали: це збірна назва для всіх інших речей, що становлять сайт, такі як зображення, музика, відео, документи Word і PDF.

Коли ви вводите веб-адресу в свій браузер (для нашої аналогії - відвідуєте магазин):

1. Браузер звертається до DNS сервера і знаходить реальну адресу сервера, на якому "живе" сайт (Ви знаходите адресу магазину).
2. Браузер посилає HTTP запит до сервера, запитуючи його відправити копію сайту для клієнта (Ви йдете в магазин і замовляєте товар). Це повідомлення і всі інші дані, що передаються між клієнтом і сервером, передаються по інтернет-з'єднання з використанням протоколу TCP / IP.

3. Якщо сервер схвалює запит клієнта, сервер відправляє клієнту статус "200 ОК", який означає: "Звичайно, ви можете подивитися на цей сайт! Ось він", а потім починає відправку файлів сайту в браузер у вигляді невеликих порцій, які називаються пакетними даними (магазин видає вам ваш товар або вам привозять його додому).
4. Браузер збирає маленькі шматки в повноцінний сайт і показує його вам (товар прибуває до ваших дверей - нові речі, приголомшливо!).

1.1.2 Типи додатків

Розробка веб-додатків не обмежується лише смартфонами або планшетами. Він призначений для роботи в будь-якому браузері, роботі на настільних комп'ютерах, ноутбуках або мобільних пристроях. Вирізняють такі типи веб-додатків:

Статичний веб-додаток. Цей тип веб-додатоків відображає дуже мало вмісту і не є досить гнучким. Зазвичай вони розробляються в HTML та CSS . Однак анімовані об'єкти, такі як банери, GIF, відео тощо, також можуть бути включені і показані в них. Вони також можуть бути розроблені за допомогою jQuery та Ajax.

Крім того, змінити вміст статичних веб-додатків непросто. Для цього спочатку потрібно завантажити HTML-код, потім змінити його і, нарешті, знову завантажити його на сервер. Ці зміни можуть бути внесені лише веб-майстром.

Динамічний веб-додаток. Динамічні веб-додатки набагато складніші на технічному рівні. Вони використовують бази даних для завантаження даних, а їх вміст оновлюється кожного разу, коли користувач звертається до них. Зазвичай вони мають панель адміністрування (називається CMS), звідки адміністратори можуть виправляти або змінювати вміст програми, включаючи текст та зображення.

Для динамічної розробки веб-додатків можна використовувати безліч різних мов програмування. JavaScript - це найпоширеніша мова, яка використовуються для цієї мети.

У такому додатку оновлення вмісту дуже просте, і сервер навіть не потребує доступу до нього, змінюючи його. Крім того, це дозволяє реалізувати безліч функцій, таких як форуми або бази даних. Дизайн - окрім вмісту - може бути змінений відповідно до уподобань адміністратора.

Інтернет-магазин або електронна комерція. Якщо веб-додаток - це інтернет-магазин чи магазин, його розробка, швидше за все, буде схожа на розвиток електронної комерції або веб - сайт електронної комерції. Цей процес розробки додатків є складнішим, оскільки він повинен включати електронні платежі за допомогою кредитних карток, PayPal або інших способів оплати. Розробник також повинен створити панель управління для адміністратора. Він буде використовуватися для перерахування нових продуктів, їх оновлення або видалення та управління замовленнями та платежами.

Веб-додаток порталу Під порталом маємо на увазі своєрідну програму, в якій ми отримуємо доступ до декількох її розділів або категорій через домашню сторінку. Ці програми можуть включати багато речей: форуми, чати, електронну пошту, браузери, області, до яких можна отримати реєстрацію, останній вміст тощо.

Анімаційний веб-додаток - неминуче пов'язаний з технологією FLASH. Цей підхід програмування дозволяє представити контент з анімованими ефектами. Це також дозволяє більш інноваційні та сучасні конструкції та є однією з найбільш широко використовуваних технологій дизайнерів та креативних директорів. Недолік, властивий розробці анімованих веб-додатків, полягає в тому, що такий вид технології не підходить для цілей розміщення в Інтернеті та оптимізації SEO, оскільки

пошукові системи не можуть правильно читати інформацію, яку вони містять.

Вміст повинен постійно оновлюватися, коли мова йде про розробку веб-додатків, тому встановлення системи управління вмістом (CMS) є серйозним варіантом. Адміністратор може використовувати цю CMS для впровадження змін та оновлень. Ці менеджери вмісту інтуїтивно зрозумілі та дуже прості в обробці.

1.2 Базові технології побудови веб-додатків

Веб-розробка має величезний набір правил та методик, про які повинен знати кожен розробник веб-сайту та веб-додатку. Якщо ви хочете, щоб веб-сайт виглядав і працював так, як ви і хотіли, вам потрібно ознайомитися з веб-технологіями, які допоможуть вам досягти поставленої мети. Розробка програми або веб-сайту зазвичай зводиться до знання трьох основних мов: JavaScript , CSS та HTML .

Оскільки комп'ютери не можуть спілкуватися між собою так, як це роблять люди, їм потрібні коди. Веб-технології - це мови розмітки та мультимедійні пакети, які комп'ютери використовують для спілкування.

При створенні веб-додатків, доволі часто використовуються API. API (інтерфейс програмування додатків) дозволяє іншим розробникам використовувати частину функцій програми без спільного використання коду. Кінцеві точки відкриваються розробниками, тоді як API може контролювати доступ за допомогою ключа API.

Дані зберігаються в структурі, що називається формат даних.

JSON - JavaScript Object Notation - це синтаксис для зберігання та обміну даними (як і XML). В даний час це найпопулярніший формат даних там.

XML - В основному використовується системами Microsoft, він раніше був найпопулярнішим форматом даних.

CSV - це формати даних комами; наприклад, дані Excel.

Кожен користувач програми називається клієнтом. Клієнтами можуть бути комп’ютери, мобільні пристрої, планшети тощо. Зазвичай кілька клієнтів взаємодіють з одним і тим же додатком, що зберігається на сервері.

Код програми зазвичай зберігається на сервері. Клієнти роблять запити до серверів. Потім сервери відповідають на ці запити після збору запитуваної інформації.

1.2.1 HTML

HTML - основна мова розмітки Всесвітньої павутини. Спочатку HTML насамперед був розроблений як мова для семантичного опису наукових документів. Однак його загальна конструкція дозволила адаптувати її протягом наступних років для опису низки інших типів документів і навіть додатків.

Документи HTML складаються з дерева елементів та тексту. Кожен елемент позначається у джерелі початковим тегом , таким як " <body>" та кінцевим тегом, таким як " </body>". Теги повинні бути вкладені таким чином, щоб усі елементи були повністю один в одному, без перекриття.

Агенти користувача HTML (наприклад, веб-браузери) потім аналізують цю розмітку, перетворюючи її в дерево DOM (Document Object Model). Дерево DOM - це представлення документа в пам'яті. Цим деревом DOM можна керувати зі скриптів на сторінці. Сценарії (як правило, в JavaScript) - це невеликі програми, які можна вбудовувати за допомогою script елемента або за допомогою атрибутів вмісту обробника подій.

Кожен елемент дерева DOM представлений об'єктом, і ці об’єкти мають API, щоб ними можна було маніпулювати. Оскільки дерева DOM використовуються як спосіб представлення документів HTML при їх обробці та поданні реалізаціями (особливо інтерактивними реалізаціями, такими як

веб-браузери). HTML-документи представляють незалежний від медіа опис інтерактивного контенту. Документи HTML можуть бути виведені на екран.

Щоб точно вплинути на те, як відбувається таке надання, автори можуть використовувати мову стилів, наприклад CSS. Коли HTML використовується для створення інтерактивних сайтів, слід бути обережним, щоб уникнути вразливості, через яку зловмисники можуть порушити цілісність самого сайту або його користувачів. Модель безпеки Інтернету базується на концепції "джерела", і, відповідно, багато потенційних атак на Інтернет пов'язані з крос-похідними діями.

Сценарії в HTML мають семантику "запуск до завершення", тобто браузер, як правило, виконує сценарій безперебійно, перш ніж робити щось інше, наприклад, запускати подальші події або продовжувати розбір документа.

З іншого боку, аналіз HTML-файлів відбувається поступово, це означає, що аналізатор може зробити паузу в будь-якій точці, щоб дозволити запуску сценаріїв. Це взагалі гарна річ, але це означає, що авторам потрібно бути обережними, щоб уникнути підключення обробників подій після того, як події могли бути звільнені.

Існує два прийоми надійної роботи: використовувати атрибути вмісту обробника подій або створити елемент та додати обробники подій у той самий сценарій. Останнє є безпечним, оскільки, як було сказано раніше, сценарії запускаються до завершення, перш ніж подальші події можуть розпочати.

1.2.2 CSS та SASS

Як ми вже пояснили в попередньому розділі, елементи HTML дозволяють розробнику розмітити документ відповідно до його структури. Наприклад, можна бути більш-менш впевненим, що зміст контейнера strong буде відображено напівжирним шрифтом. Цілком можна довіряти і тому

факту, що більшість браузерів відобразить вміст контейнера h1 великим шрифтом, крупніше, ніж p , і крупніше, ніж h2 . Однак крім віри і надії на це, засобів контролю за зовнішнім виглядом тексту у нас просто немає. CSS все змінює.

Таблиці стилів складаються з повторюваних блоків правил. Ці блоки називаються – CSS правило. Правило – це виклик стилістичного параметра одного елемента або декількох. Таблиця стилів це набір з одного або більше правил, які додаються до HTML-документу. Правило складається з двох частин: Селектора - частини перед лівої фігурною дужкою; Оголошення - частини всередині фігурних дужок. Селектор це ланка, сполучна HTML-документ і стиль. Воно встановлює на які елементи впливає оголошення. Оголошення це частина правила, яка визначає ефект. Оголошення має дві частини, розділені двокрапкою: Властивість - частина перед двокрапкою; Значення - частина після двокрапки. Властивість — це якість або характеристика, якими щось володіє. Значення — це точна специфікація властивості.

Щоб змусити будь-яку таблицю стилів вплинути на документ HTML, її потрібно "приклеїти" до нього. Тобто, таблиця і документ повинні бути об'єднані, щоб спрацювати разом і представити документ. Це можна зробити будь-яким з чотирьох способів:

- Застосувати базову, внутрішню таблицю стилів, до документа, використовуючи тег style .
- Застосувати таблицю стилів до окремого тегу, використовуючи атрибут style .
- Прив'язати зовнішню таблицю стилів до документа, використовуючи елемент link .
- Імпортувати написані таблиці стилів, використовуючи запис CSS @import.

Sass - мова таблиці стилів, яка компілюється в CSS. Це дозволяє використовувати змінні , вкладені правила , мікси, функції та інше, все з повністю сумісним CSS синтаксисом. Sass допомагає підтримувати великі таблиці стилів добре організованими та дозволяє легко ділитися дизайном в межах та між проектами.

Сам CSS може бути цікавим, але таблиці стилів стають все більшими, складнішими та складнішими в обслуговуванні. Тут може допомогти препроцесор. Sass дозволяє використовувати функції, які ще не існують у CSS, як-от змінні, вкладені, міксин, успадкування та інші. Допомагає зменшити повторення CSS та економить час. Це більш стабільна і потужна мова розширення CSS, яка чітко та структурно описує стиль документа.

Для можливості використовувати Sass, потрібно попередньо оброблений файл Sass і зберегти його як звичайний CSS- файл, який ви можете використовувати на своєму веб-сайті.

SCSS синтаксис використовує розширення файлу .scss. З кількома невеликими винятками. Scss – це супер набір css, що по суті означає, все що працює в CSS добре працює в SCSS. Через свою схожість з CSS , це найпростіший синтаксис та найпопулярніший, до якого вже всі звикли.

```
@mixin button-base() {
  @include typography(button);
  @include ripple-surface;
  @include ripple-radius-bounded;

  display: inline-flex;
  position: relative;
  height: $button-height;
  border: none;
  vertical-align: middle;

  &:hover { cursor: pointer; }

  &:disabled {
    color: $mdc-button-disabled-ink-
color;
    cursor: default;
    pointer-events: none;
  }
}
```

Рис. 2 Приклад коду Scss

Спочатку він був розроблений Hampton Catlin та розроблений Natalie Weizenbaum у 2006 році. Пізніше Weizenbaum та Chris Eppstein використали свою первісну версію для розширення Sass за допомогою SassScript.

SASS - це мова попередньої обробки, яка надає відступний синтаксис (власний синтаксис) для CSS. Він надає деякі функції, які використовуються для створення таблиць стилів, що дозволяє більш ефективно писати код і простий у обслуговуванні. Це супер набір CSS, а це означає, що він містить усі можливості CSS і є попереднім процесором з відкритим кодом, закодованим у Ruby . Він забезпечує стиль документа в хорошому, структурованому форматі, ніж плоский CSS. Він використовує методи, що повторно використовуються, логічні твердження та деякі вбудовані функції, такі як маніпуляція кольором, математика та списки параметрів.

Особливості SASS

- Він більш стабільний, потужний та сумісний з версіями CSS.
- Це супер набір CSS і заснований на JavaScript.

- Він відомий як синтаксичний цукор для CSS, а це означає, що він полегшує користувачеві зрозуміти чи висловити речі більш чітко.
- Він використовує власний синтаксис і компілюється для читання CSS.
- Ви можете легко записати CSS за меншим кодом за менший час.
- Це попередній процесор з відкритим кодом, який інтерпретується в CSS.

Переваги SASS

- Це дозволяє записати чистий CSS у програмування.
- Це допомагає швидко писати CSS.
- Це набір CSS, який допомагає дизайнерам та розробникам працювати ефективніше та швидше.
- Оскільки Sass сумісний з усіма версіями CSS, ми можемо використовувати будь-які наявні бібліотеки CSS.
- Можна використовувати вкладений синтаксис та корисні функції, такі як маніпуляція кольором, математика та інші значення.

Недоліки SASS

- Потрібен час, щоб розробник дізнався нові функції, наявні в цьому попередньому процесорі.
- Якщо багато людей працюють на одному веб-сайті, то слід використовувати той самий препроцесор. Деякі люди використовують Sass, а деякі використовують CSS для редагування файлів безпосередньо. Тому працювати на сайті стає важко.
- Є ймовірність втратити переваги вбудованого інспектора елементів браузера.

1.2.3 JavaScript та JQuery

Згідно з щорічним опитуванням StackOverflow, найпопулярнішою мовою програмування є JavaScript, 62,5% респондентів заявляють, що ним користуються.

JavaScript – це найпоширеніший сценарій мови ECMAscript, який розробляється компанією Ecma , велика частина API, доступних у браузерах, була визначена на W3C.

Сценарій - це програмний код, який не потребує попередньої обробки (наприклад, компілювання) перед запуском. У контексті веб-браузера сценарій зазвичай стосується програмного коду, написаного на JavaScript, який виконується браузером під час завантаження сторінки або у відповідь на подію, викликану користувачем. Сценарій може зробити веб-сторінки більш динамічними. Наприклад, не завантажуючи нову версію сторінки, вона може дозволяти змінювати вміст цієї сторінки або дозволяти додавати вміст на цю сторінку або надсилати її. Перший отримав назву DHTML (Динамічний HTML), а другий - AJAX (Асинхронний JavaScript та XML). Крім цього, сценарії все частіше дозволяють розробникам створити міст між браузером і платформою, на якій він працює, що дозволяє, наприклад, створювати веб-сторінки, які містять інформацію з середовища користувача, наприклад, поточне місцезнаходження, реквізити адресної книги, тощо. Ця додаткова інтерактивність змушує веб-сторінки вести себе як традиційне програмне забезпечення. Ці веб-сторінки часто називають веб-програмами або веб-додатки і можуть бути доступні безпосередньо у веб-переглядачі у вигляді веб-сторінки, або можуть бути упаковані та розповсюджені у вигляді віджетів.

JavaScript - це об'єктно-орієнтована мова програмування для виконання обчислень та маніпулювання обчислювальними об'єктами в хост-середовищі.

Мова сценаріїв - мова програмування, яка використовується для маніпулювання, налаштування та автоматизації можливостей існуючої системи. У таких системах корисна функціональність вже доступна через користувальницький інтерфейс, а мова сценаріїв - це механізм викриття цієї функціональності програмним управлінням. Таким чином, кажуть, що існуюча система забезпечує вузлове середовище об'єктів та об'єктів, що

доповнює можливості мови сценаріїв. Сценарна мова призначена для використання як професійними, так і непрофесійними програмістами.

Спочатку JavaScript був розроблений як мова веб-скриптів , що забезпечує механізм оживлення веб-сторінок у браузерах та проведення обчислень на сервері як частини веб-архітектури клієнт-сервер. JavaScript може надавати основні можливості скриптування для різних середовищ хоста. Деякі засоби JavaScript схожі на ті, що використовуються в інших мовах програмування.

Веб-браузер забезпечує середовище хоста JavaScript для обчислення на стороні клієнта, включаючи, наприклад, об'єкти, що представляють вікна, меню, спливаючі вікна, діалогові вікна, текстові області, анкери, кадри, історію, файли cookie та введення / виведення. Крім того, хост-середовище забезпечує засіб приєднання сценарію коду до таких подій, як зміна фокуса, завантаження сторінки та зображень, вивантаження, помилка та переривання, вибір, подання форми та дії миші. Код сценарію відображається в HTML, а відображається сторінка - це поєднання елементів інтерфейсу користувача та фіксованого та обчисленого тексту та зображень. Код сценарію реагує на взаємодію користувачів і немає необхідності в основній програмі.

Веб-сервер забезпечує інше середовище хоста для обчислення на стороні сервера, включаючи об'єкти, що представляють запити, клієнтів та файли; механізми блокування та обміну даними. При спільному використанні сценаріїв на стороні браузера та на сервері можна розподілити обчислення між клієнтом і сервером, надаючи індивідуальний користувальницький інтерфейс для веб-додатків.

JavaScript базується на об'єктах: основна мова та засоби хосту забезпечуються об'єктами, а програма JavaScript - це кластер об'єктів, що спілкуються. JavaScript об'єкт являє собою набір властивостей, кожен з нуля або більше атрибутів , які визначають , як кожна властивість може бути використана. Властивості - це контейнери, що містять інші об'єкти, примітивні значення або функції.

JavaScript також визначає набір вбудованих операторів . Оператори JavaScript включають різні одинарні операції, мультиплікативні оператори, оператори адитивних операцій, оператори зсувної передачі, реляційні оператори, оператори рівності, бінарні побітові оператори, бінарні логічні оператори, оператори присвоєння та оператор коми.

Синтаксис JavaScript навмисно нагадує синтаксис Java. Синтаксис JavaScript легший, щоб він міг слугувати простою у використанні мовою сценаріїв. Наприклад, змінній не потрібно оголошувати її тип, а також типи, пов'язані з властивостями, а визначеним функціям не потрібно, щоб їх декларації відображалися текстуально перед викликами до них.

jQuery - це швидка, невелика та багатофункціональна бібліотека JavaScript. Завдяки простому у користуванні API, який працює у безлічі браузерів, такі речі, як обробка та маніпулювання документами HTML, обробка подій, анімація та Ajax, набагато простіші. Завдяки поєднанню універсальності та розширюваності, jQuery змінив спосіб, по якому мільйони людей пишуть JavaScript.

Бібліотека jQuery розкриває свої методи та властивості за допомогою двох властивостей windowоб'єкта під назвою jQueryута $. $це просто псевдонім для, jQueryі його часто використовують, оскільки писати коротше і швидше. Доступ до всієї можливості jQuery здійснюється через JavaScript, тому для розуміння, структуризації та налагодження коду важливе значення має розуміння JavaScript.

1.2.4 AJAX

AJAX - це мрія розробника, адже він надає можливість:

- Оновити веб-сторінку, не завантажуючи її повністю
- Надсилати запит даних на сервер - після завантаження сторінки
- Отримувати дані від сервера - після завантаження сторінки
- Надсилати дані на сервер - у фоновому режимі

Завантаження частини даних веб-сайту за допомогою JavaScript - запитів відоме як AJAX , для асинхронного JavaScript та XML.

Ajax - це об'єктно-орієнтований модуль, який забезпечує унікальний механізм використання асинхронного коду з HTML-сторінок, що підтримують JavaScript. Ajax забороняє користувачеві писати обширний JavaScript, за винятком асоціації експортованого методу з визначеною документом подією (наприклад, onClick, onKeyUp тощо). Ajax також добре поєднується з HTML, що містить складніший JavaScript.

Ajax підтримує методи, що повертають окремі результати або кілька результатів на веб-сторінку, а також підтримує повернення значень для декількох елементів DIV на сторінці HTML.

Використовуючи Ajax, URL-адресу запита HTTP GET / POST автоматично генерується на основі макета HTML та подій, і сторінка потім динамічно оновлюється.

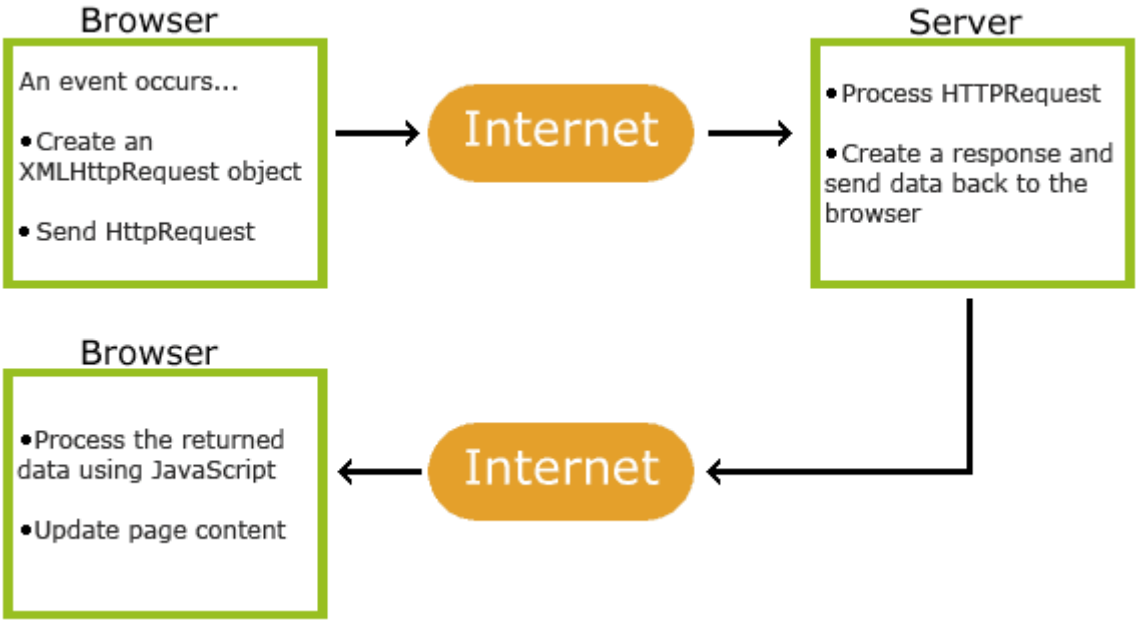


Рис. 3 Процес оновлення інформації

1. Подія відбувається на веб-сторінці (сторінка завантажується, натискається кнопка)
2. Об'єкт XMLHttpRequest створюється JavaScript
3. Об'єкт XMLHttpRequest надсилає запит на веб-сервер

4. Сервер обробляє запит

5. Сервер відправляє відповідь назад на веб-сторінку

6. Відповідь читається JavaScript

7. Правильні дії (наприклад, оновлення сторінки) виконуються JavaScript

1.2.5 Node.js

Node.js - середовище виконання JavaScript. Середовище виконання Node.js включає все, що потрібно для виконання програми, написаної на JavaScript.

Node.js з'явився тоді, коли оригінальні розробники JavaScript розширили його з чогось, що ви могли запустити лише в браузері, до того, що ви могли працювати на вашій машині як окремий додаток. З’явилося можливість щось створювати набагато більше з JavaScript, ніж просто робити веб-сайти інтерактивними. Тепер JavaScript має можливість робити те, що можуть робити інші мови сценаріїв.

Node.js - це серверна платформа, побудована на JavaScript Chrome Engine (V8 Engine). Node.js був розроблений Райаном Далом у 2009 році. Цей двигун бере код JavaScript та перетворює його в більш швидкий машинний код. Машинний код - це код низького рівня, який може працювати на комп'ютері без необхідності його першої інтерпретації. Node.js використовує подію, що не блокує модель вводу / виводу, що робить її легкою та ефективною. Введення / виведення відноситься до вводу / виводу. Це може бути все, починаючи від читання / запису локальних файлів до подання HTTP-запиту до API. Введення / виведення потребує часу і, отже, блокує інші функції.

Нижче наведено деякі важливі функції, які роблять Node.js першим вибором архітекторів програмного забезпечення.

- Асинхронний та керований подіями - всі API бібліотеки Node.js асинхронні, тобто не блокують. Це по суті означає, що сервер на базі Node.js ніколи не чекає повернення даних API. Після виклику сервера

переходить до наступного API, а механізм сповіщень про події Node.js допомагає серверу отримати відповідь від попереднього виклику API.

- Дуже швидко - побудована на V8 JavaScript Engine Engine Google, бібліотека Node.js дуже швидка у виконанні коду.
- Одинарна нитка, але дуже масштабована - Node.js використовує єдину модель потоку з циклічною подією. Механізм подій допомагає серверу реагувати не блокуючим способом і робить сервер високо масштабованим на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує одну потокову програму, і ця сама програма може надавати послуги набагато більшій кількості запитів, ніж традиційні сервери, такі як Apache HTTP Server.
- Без буферизації - програми Node.js ніколи не буферують будь-які дані. Ці програми просто виводять дані шматками.
- Ліцензія - Node.js випускається під ліцензією MIT

NPM - це бібліотеки, створені спільнотою, яка вирішує більшість загальних проблем. NPM – це менеджер пакетів вузлів. Він має пакети, які використовуються у додатках, щоб зробити розробку швидшою та ефективнішою.

Модуль вузла - це блок коду багаторазового використання, існування якого не впливає випадковим чином на інший код.

Кожен хто користується Node.js має можливість написати власні модулі та використовувати їх у різних додатках. Node.js має набір вбудованих модулів, якими можно користуватися без подальшої установки.

V8 - це турбо-двигун JavaScript який використовує в своїй основі мову програмування C ++. V8 реалізує сценарій під назвою ECMAScript, як зазначено в ECMA-262. ECMAScript був створений компанією Ecma International для стандартизації JavaScript. V8 може працювати окремо або вбудовуватися в будь-яку програму на C ++. Він має гачки, які дозволяють

створити власний код C ++, який ви можете зробити доступним для JavaScript. Це по суті дозволяє додавати функції в JavaScript, встановлюючи V8 у свій код C ++, щоб ваш код C ++ розумів більше, ніж те, що іншим чином визначає стандарт ECMAScript. Окрім V8 від Chrome, існує багато різних двигунів виконання JavaScript, такі як SpiderMonkey від Mozilla, Chakra від Microsoft тощо.

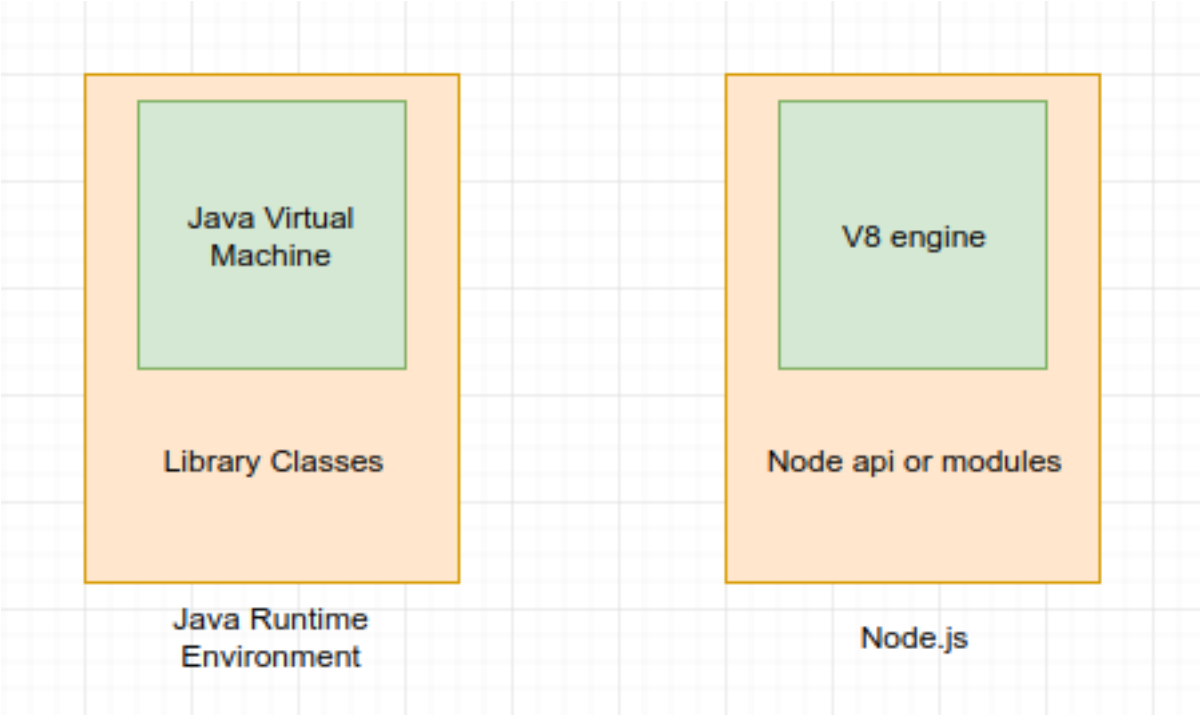


Рис. 4 Двигун Java та Node.js

Події – це дії, що виконуються у додатку та на які ми можемо відповісти.

У Node існує два типи подій:

- Системні події: ядро C ++ з бібліотеки під назвою libuv. (Наприклад, закінчив читання файлу).
- Спеціальні події: ядро JavaScript.

Node.js дає можливість створювати програмні комплекси різних типів:

1. Односторінкові програми (SPA): це веб-додатки, які працюють у браузері і для користування ними не потрібно перезавантажувати сторінку щоразу коли ви отримуєте нові дані. Деякі приклади SPA-програм включають додатки для соціальних мереж, програми

електронної пошти чи карти, онлайн-інструменти для малювання тексту та ін.

2. Програми в реальному часі (RTA): це веб-додатки, які дозволяють користувачам отримувати інформацію, як тільки вона публікується автором і головне не вимагають, щоб користувач (або програмне забезпечення) періодично перевіряли джерело на оновлення. Деякі приклади RTA включають додатки для обміну миттєвими повідомленнями або чати, ігри які можна грати в браузері, документи в режимі онлайн, спільне зберігання, програми відеоконференцій тощо.
3. Додатки для потокової передачі даних : це програми або служби, які надсилають дані по мірі їх надходження або створення, тримаючи з'єднання відкритим, щоб продовжувати завантажувати додаткові дані за потребою. Деякі приклади включають програми для відео та аудіо потоку.
4. API REST : це інтерфейси, які надають дані для взаємодії з чужим веб-додатком. Наприклад, служба API API може надати дату та час проведення певної події, які можуть бути використані чужим веб-сайтом.
5. Програми, надані на сервері (SSR): Ці веб-програми можуть працювати як на клієнті – браузері, так і на сервері, що дозволяє динамічно відображати сторінки, які вже відомі, і швидко захоплювати вміст. Їх часто називають «ізоморфними» або «універсальними» додатками. SSR використовують методи SPA, оскільки їх не потрібно кожного разу перезавантажувати.
6. Програми командного рядка: вони дозволяють автоматизувати задачі які постійно повторюються, а потім поділитися цим інструментом по всій екосистемі Node.js. Прикладом інструменту командного рядка є cURL, який є URL-адресою клієнта і використовується для завантаження вмісту з Інтернет-адреси. cURL часто використовується

для встановлення додаткових модулів або додатків, таких як Node.js або менеджер версій Node.js.

7. Апаратне програмування : хоча Node.js не настільки популярний, як веб-додатки, Node.js. Проте зростає популярність у використання в мережі Інтернет, сервісів для збирання даних з датчиків, маяків, передавачів, двигунів, тощо, що генерує велику кількість даних. Node.js може ввімкнути збір даних, проаналізувати ці дані, спілкуючись з пристроєм та сервером, і вже вживати певні дії на основі аналізу. NPM містить понад 80 пакетів для контролерів Arduino, різних датчиків та пристроїв Bluetooth.

1.2.6 React та JSX

React - це бібліотека JavaScript для створення інтерфейсів користувача. React – це проект створений для кращої реалізація Facebook та Instagram. Який забезпечує частину парадигми розвитку MVC (модель-перегляд-контролер) "перегляд". Він був створений для того, щоб розробники могли створювати великі програми з даними, які змінюються. Робота над React розпочалася приблизно в кінці 2011 року, а сама технологія були відкрита у 2012. React дозволяє створювати компоненти інтерфейсу замість шаблонів.

React базується на ідеї декларативно вказувати користувальницькі інтерфейси поверх моделі даних, де інтерфейси користувачів автоматично підтримують синхронізацію з базовими даними. Він робить безболісним створення інтерактивних інтерфейсів користувача. Створюючи прості подання для кожного стану у програмі, завдяки цьому він ефективно оновлює та надає лише потрібні компоненти, коли дані зміняються. Декларацiйні представлення роблять код більш передбачуваним, простішим для розуміння та простішим для налагодження.

React використовує JavaScript, а не HTML для створення інтерфейсів користувача, наводячи гнучкість як причину такого дизайнерського рішення. Він створює інкапсульовані компоненти, що управляють власним станом, а

потім компонують їх, щоб скласти складні інтерфейси користувача. Оскільки логіка компонентів написана в JavaScript замість шаблонів, маємо можливість легко передавати великий об’єм даних через додаток і не підтримувати стан DOM.

Популярність React обумовлена використанням віртуального DOM, вбудовування HTML в JavaScript з JSX, організація інтерфейсу користувача в компоненти. Він принципово змінює погляд, з яким всі розробники працювали роками.

З тих пір, як у веб-браузерах була запроваджена і широко прийнята модель об’єкта документа (DOM), веб-розробники зустрічалися з одною проблемою - DOM повільний. Адже, внесення змін до DOM відбувається повільно через перефарбовування та поповнення, виникають витоки пам’яті, через не відслідковування посилань на вузли. Постійно потрібно переконуватися, що ви знімаєте та повторно приєднуєте обробників подій у потрібних місцях. Це означає, що вартість оновлення DOM досить висока, тому користь яку ви отримаєте повинно бути достатньо високою, щоб виправдати витрати.

Тому React був створений для вирішення цих проблем. Тому-що він керую вашими обробниками подій для вас, гарантуючи, що вони прикріплені та від’єднані в потрібний час та на правильних вузлах. Це створює та ефективно знищує структури DOM. Також використовується віртуальний DOM, який відрізняється, щоб визначити, які частини компонента змінилися, і лише оновлює ці частини. Усі ці технічні рішення змінюють старі погляди на оновлення DOM.

З React не потрібно замислюватися про повторне відображення всієї сторінки, тому що не потрібно рендерити все. Необхідно лише рендерити шматки, які потребують змін. Логіка React повертає нас до ментальної модель, як і в традиційних програмах, призначених лише для сервера.

Сторінка надається, деякі зміни запитуються, потім відображається сторінка з тими змінами. Єдина відмінність - це все може статися на стороні клієнта.

Одна із можливостей React - це використання синтаксису схожого на HTML, який називається - JSX . JSX не вимагає використання React, але він робить код більш читабельним, і при його написанні він виглядає як написання HTML.

JSX - це синтаксис, подібний XML / HTML, на далі тільки HTML, який використовується React, який розширює ECMAScript, щоб текст, подібний XML/HTML, міг співіснувати з кодом JavaScript/React. Синтаксис призначений для використання препроцесорами (тобто транспіляторами, такими як Babel) для перетворення тексту, схожого на HTML, знайденого у файлах JavaScript, у стандартні об’єкти JavaScript, які механізм JavaScript буде аналізувати.

В основному, використовуючи JSX, надається можливість створювати лаконічні HTML-подібні структури (наприклад, деревоподібні структури DOM) у той самий файл, де використовується код JavaScript, тоді Babel перетворить ці вирази у фактичний код JavaScript. На відміну від минулого, замість того, щоб JavaScript вводити в HTML, JSX дозволяє нам вводити HTML в JavaScript.

ВИСНОВОК ДО РОЗДІЛУ 1

В даному розділі було розглянуто основні принципи побудови веб-додатків. Описано завдяки чому вони працюють та маюсь доступ в мережі Інтернет. Приведено типи веб-додатків які існують та створюються на даний момент. Досліджено архітектуру типового клієнт-серверного додатку. Описано основні технології для розробки під веб та їх популярні надбудови. Розглянуто популярні та доцільні методи для створення клієнтської та серверної частини додатку.

Визначено, що JQuery, Node.js та React разом з JSX являються похідними від JavaScript. Ці бібліотеки мови програмування JavaScript, використовуються для спрощення процесу та складності написання програмного забезпечення на даній мові. Ці бібліотеки створюються для швидкого виконання та написання певних постійно використовуємих конструкцій. Вони розроблені для вирішення окремих задач, котрі виконуються швидше та глибше ніж у оригінальній мові. Встановлено, що певні методи створення клієнтської частини додатку, можуть та краще вже використовувати на серверній частині програми. Це дає змогу спростити та пришвидшити процес відображення динамічних компонентів інтерфейсу.

Не існує обов’язкових методів або технологій для створення веб-сайтів або веб-додатків. Кожен розробник має можливість користуватися будь якими технологіями та об’єднувати їх на свій розсуд.

РОЗДІЛ 2 АНАЛІЗ АРХІТЕКТУРНИХ ШАБЛОНІВ ТА ПРОЕКТУВАННЯ МАЙБУТНЬОГО ВЕБ-ДОДАТКУ

2.1.Проблема вибору архітектури веб-додатку

2.1.1. Види архітектурних шаблонів

До того, як перейти до безпосередньої проектування та створення програмного забезпечення, необхідно визначити доречний архітектурний шаблон, який надає бажану якість і функціонал. Тому необхідно розібратися в нюансах різних архітектурних шаблонів ще до того, як застосувати їх до свого додатку.

Архітектурний шаблон - це загальне і повторюване рішення для вирішення проблеми архітектури додатків в межах заданого контексту. Архітектурні шаблони схожі з шаблонами програмного дизайну, проте мають більш широке охоплення.

Багаторівневий шаблон

Даний шаблон використовується для структурування програм, які можна розкласти на групи деяких підзавдань, які перебувають на певних рівнях абстракції. Кожен шар надає служби для наступного, більш високого шару.

Найчастіше в загальних інформаційних системах зустрічаються наступні 4 шару:

- Шар уявлення (також відомий як шар призначеного для користувача інтерфейсу)
- Шар додатки (також відомий як шар сервісу)
- Шар бізнес-логіки (також відомий як рівень предметної області)
- Шар доступу до даних (також відомий як рівень зберігання даних)

Використання:

- Загальні десктопні програми.

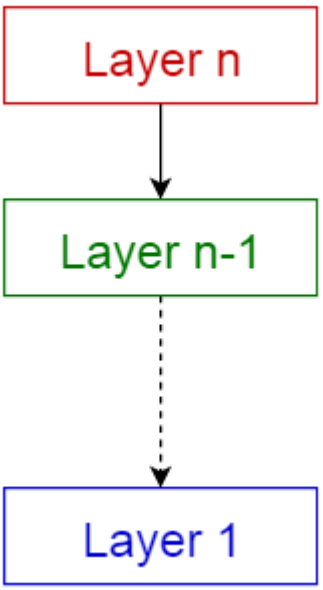
- Веб-додатки e-commerce.

Плюси:

- Одним низьким шаром можуть користуватися різні верстви вищого рангу.
- Шари спрощують стандартизацію, тому що ми чітко визначаємо рівні.
- Зміни вносяться всередині якогось одного шару, при цьому інші шари залишаються незмінними.

Мінуси:

- Не універсальний.
- У ряді ситуацій можливий пропуск деяких рівнів.



Клієнт-серверний шаблон

Даний шаблон складається з двох частин: сервера і безлічі клієнтів . Серверний компонент надає служби клієнтським компонентів. Клієнти запитують послуги у сервера, а він, у свою чергу, надає ці самі послуги клієнтам. Більш того, сервер продовжує «підслуховувати» клієнтські запити.

Використання:

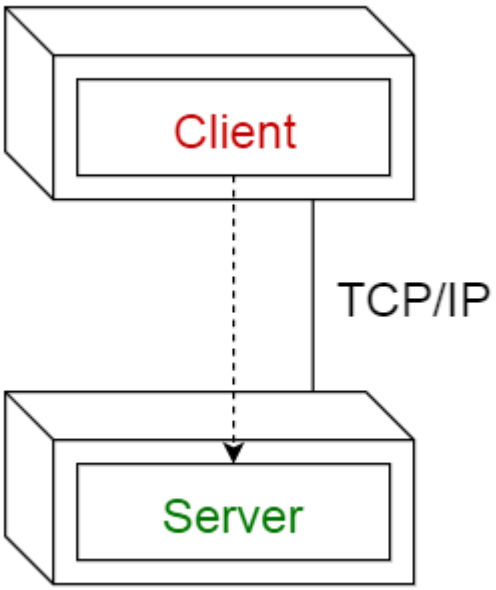
- Онлайн додатки (електронна пошта, спільний доступ до документів, банківські послуги).

Плюси:

- Підходить для моделювання набір служб, які зможуть запитувати клієнти.

Мінуси:

- Запити зазвичай виконуються в окремих потоках на сервері.
- Взаємодія між процесами підвищує ресурсовитратність, тому що різні клієнти мають різне уявлення.



Шаблон «ведучий-ведений»

У цьому шаблоні також задіяні два учасники - ведучий і ведені . Ведучий компонент розподіляє завдання серед ідентичних ведених компонентів і обчислює підсумковий результат на підставі результатів, отриманих від своїх «підлеглих».

Використання:

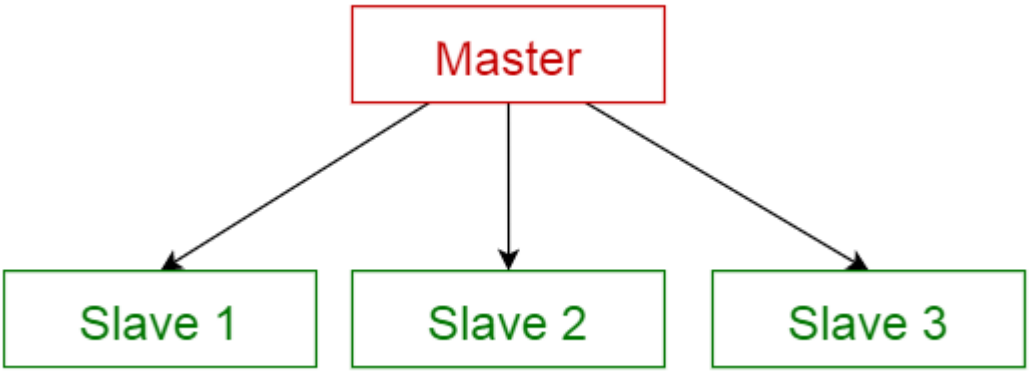
- У реплікації баз даних. Там головна БД вважається авторитетним джерелом, а підлеглі бази з нею синхронізовані.
- Периферійні пристрої, підключені до шини в комп'ютері (ведучі та ведені пристрої).

Плюси:

- Точність, тому що виконання служби делегується різними веденим з різною реалізацією.

Мінуси:

- Всі відомі ізольовані, у них відсутній загальний стан.
- Період очікування в комунікації «ведучий-ведений» - це значний мінус. Наприклад, в системах реального часу.
- Підходить тільки для тих проблем, вирішення яких можна розкласти на частини.



Шаблони «канали і фільтри»

Цей шаблон підходить для систем, які виробляють і обробляють потоки даних. Кожен етап обробки відбувається всередині якогось компонента фільтра . Дані для обробки передаються через канали . Ці канали можна використовувати для буферизації або синхронізації даних.

Використання:

- Компілятори. Послідовні фільтри виконують лексичний, синтаксичний, семантичний аналіз і створення коду.
- Робочі процеси в біоінформатики.

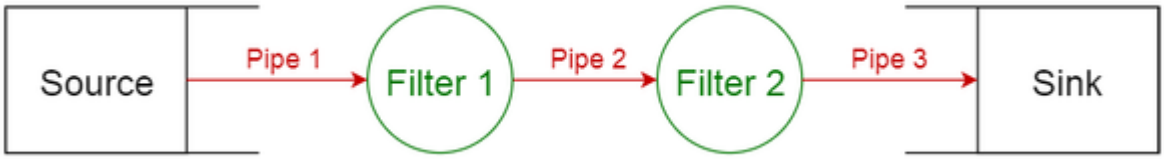
Плюси:

- Можуть реалізовувати паралельні процеси, коли вхід і вихід складається з потоків, а фільтри починають обчислення після отримання даних.
- Просте додавання фільтрів. Систему можна легко розширити.

- Фільтри підходять для повторного використання. Можуть вибудовувати різні конвеєри, створюючи всілякі комбінації існуючого набору фільтрів.

  Мінуси:

- Ефективність знижується через самих повільних процесів фільтрації.
- При переході від одного фільтра до іншого виконується трансформація даних, яка веде до підвищеного споживання ресурсів.



Шаблон «Посередник»

Даний шаблон потрібен для структуризації розподілених систем з незв'язними компонентами. Ці компоненти можуть взаємодіяти один з одним через віддалений виклик служби. Компонент посередник відповідає за координацію взаємодії компонентів .

Сервер розміщує свої можливості (служби та характеристики) у посередника (брокера). Клієнт запитує послугу у брокера. Потім брокер перенаправляє клієнта до підходящої службі зі свого реєстру.

Використання:

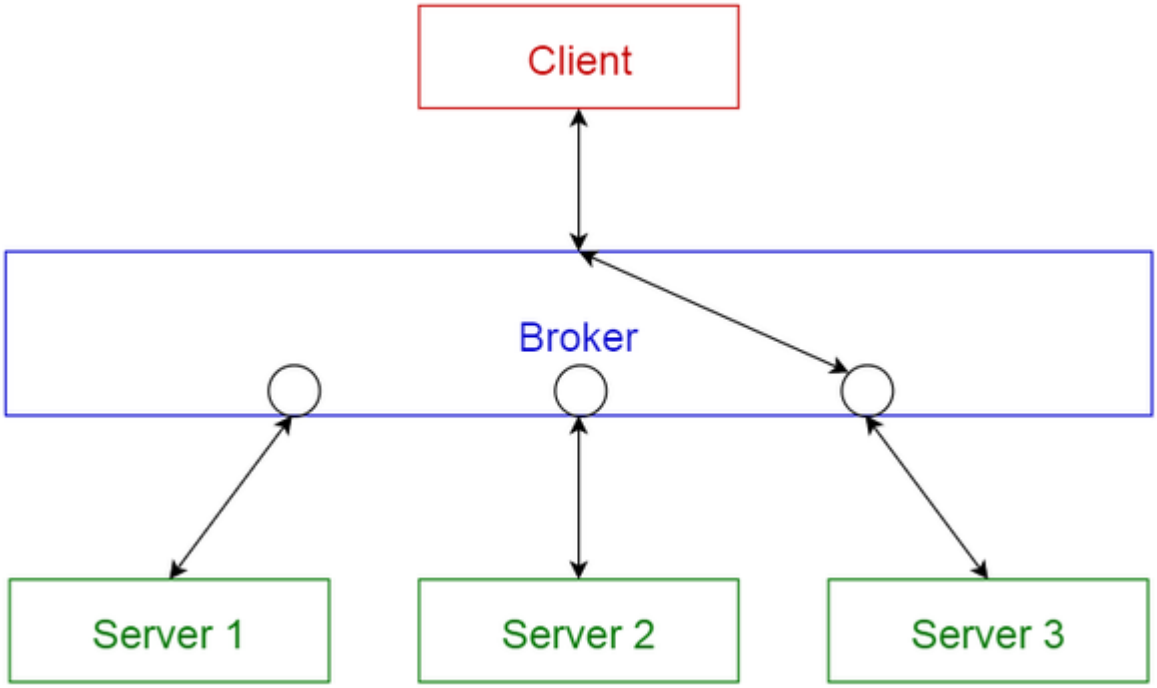
- Брокери повідомлень за типом Apache ActiveMQ , Apache Kafka , RabbitMQ і JBoss Messaging .

  Плюси:

- Можливо динамічна зміна, додавання, видалення і переміщення об'єктів. Цей шаблон робить процес розподілу прозорим для розробника.

  Мінуси:

- Необхідна стандартизація описів служб.



Одноранговий шаблон

В даному шаблоні існують окремі компоненти, так звані «піри». Бенкети можуть виступати в ролі як клієнта, що запитує послуги від інших рівноправних учасників (пірів), так і сервера, що надає послуги іншим бенкетів. Бенкет може бути клієнтом або сервером, або всім відразу, а також здатний з часом динамічно змінювати свою роль.

Використання:

- Файлообмінні мережі (Gnutella і G2)
- Мультимедіа протоколи (P2PTV і PDTP)
- Пропрієтарні мультимедійні додатки (як той же Spotify).

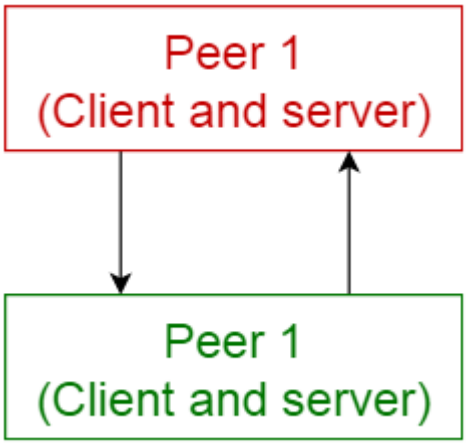
Плюси:

- Підтримує децентралізовані обчислення. Вкрай стійкий до збоїв в будь-якому вузлі.
- Висока масштабованість по частині ресурсної та обчислювальної потужності.

Мінуси:

- Відсутня гарантія якості служб, тому що вузли кооперуються стихійно.
- Важко гарантувати захищеність.

- Продуктивність залежить від кількості вузлів.



Шина подій

Цей шаблон, в основному, взаємодіє з подіями і складається з 4 основних компонентів: джерело події, прослуховувач події, канал і шина подій. Джерела розміщують повідомлення для певних каналів на шині подій. Прослуховувачі підписуються на певні канали. Прослуховувачі отримують повідомлення про появу повідомлень, розміщених на каналах з їх підписки.

Використання:

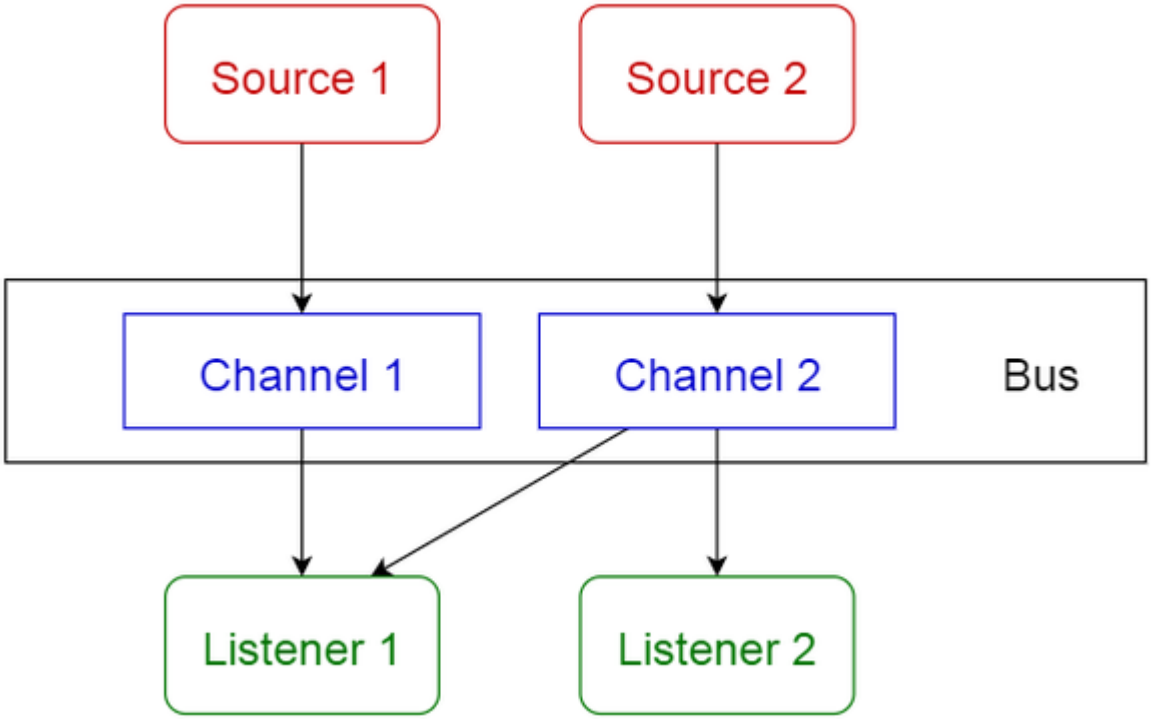
- Розробки на Android
- Сервіси повідомлень

Плюси:

- Просте додавання нових передплатників, видавців і зв'язків. Добре зарекомендував себе для сильно розподілених додатків.

Мінуси:

- Проблема з масштабністю, тому що всі повідомлення проходять через одну шину подій.



Шаблон «модель-вид-контролер» - MVC

Цей шаблон також відомий як MVC-шаблон. Він розділяє інтерактивні прикладні програми на 3 частини:

1. модель - містить ключові дані і функціонал;
2. уявлення - показує інформацію користувачеві (можна задавати більш єдиної думки);
3. контролер - займається обробкою даних від користувача.

Це робиться з метою розмежування внутрішнього представлення інформації від способів її подання і прийняття від користувача. Дана схема ізолює компоненти і дозволяє ефективно реалізувати повторне використання коду.

Використання:

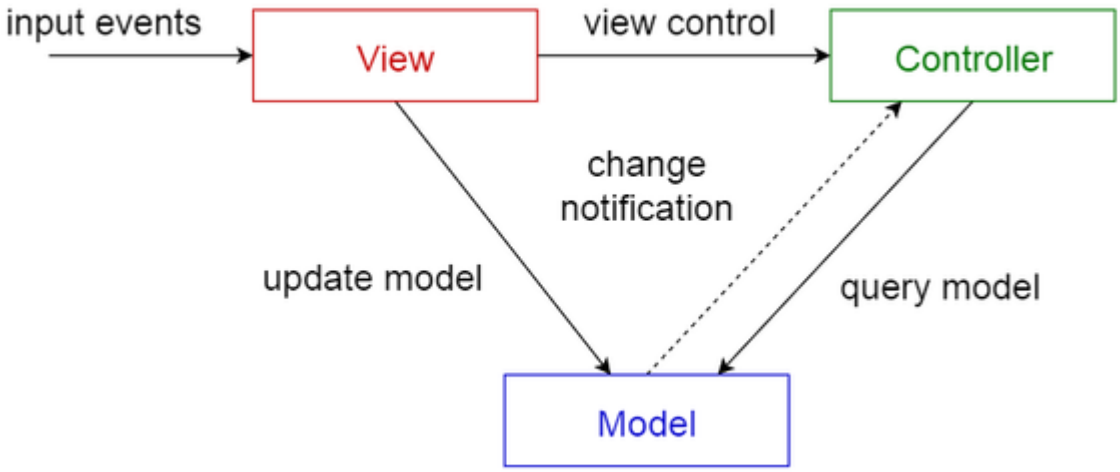
- Архітектура WWW-додатків, написаних на основних мовах програмування.
- Веб-фреймворки (наприклад, Django і Rails)

Плюси:

- Спрощує створення різних уявлень однієї і тієї ж моделі; їх можна включити або відключити на етапі виконання.

Мінуси:

- Зростає складність алгоритму. Може привести до багатьох непотрібним коректувань дій користувачів.



Шаблон «дошка»

Такий шаблон підходить для проблем, для яких відсутні чіткі детерміновані рішення. Шаблон «Дошка» складається з 3 головних компонентів:

- Дошка - це структурована глобальна пам'ять, яка містить об'єкти з простору можливих рішень;
- Джерело знань - спеціалізовані модулі зі своїм власним уявленням;
- Компоненти керування - вибирає, налаштовує і виконує модулі.

Всі компоненти мають доступ до дошки. Компоненти можуть виробляти нові об'єкти даних, які додаються до дошки. Компоненти шукають на дошці конкретні види даних. Одним із способів пошуку є зіставлення шаблонів з існуючим джерелом знань.

Використання:

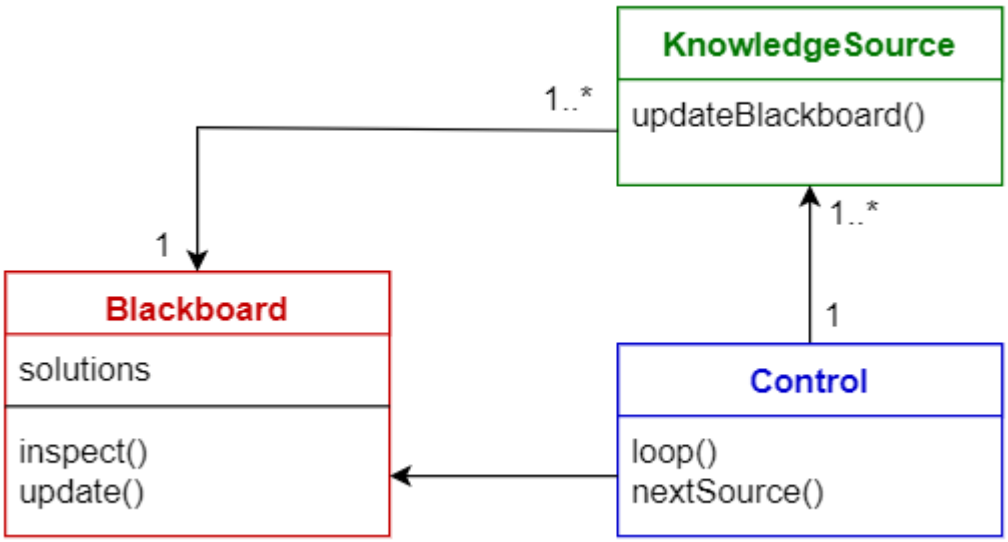
- Розпізнавання мови;
- Ідентифікація і відстеження транспортних засобів;
- Інтерпретація сигналів Sonar.

Плюси:

- Легке додавання нових додатків.
- Можна без праці розширювати структури простору даних.

Мінуси:

- Редагувати структури даних дійсно важко, тому що такі зміни зачіпають всі додатки.
- Можуть знадобитися синхронізація і управління доступом.



Шаблон «інтерпретатор»

Він підходить для розробки компонента, який повинен інтерпретувати програми, написані на спеціальній мові програмування. В основному, там розписано, як обчислювати рядки (інакше кажучи: «пропозиції» або «вираження»), написані на якомусь певному мові програмування. Суть в тому, щоб привласнити клас кожному символу мови.

Використання:

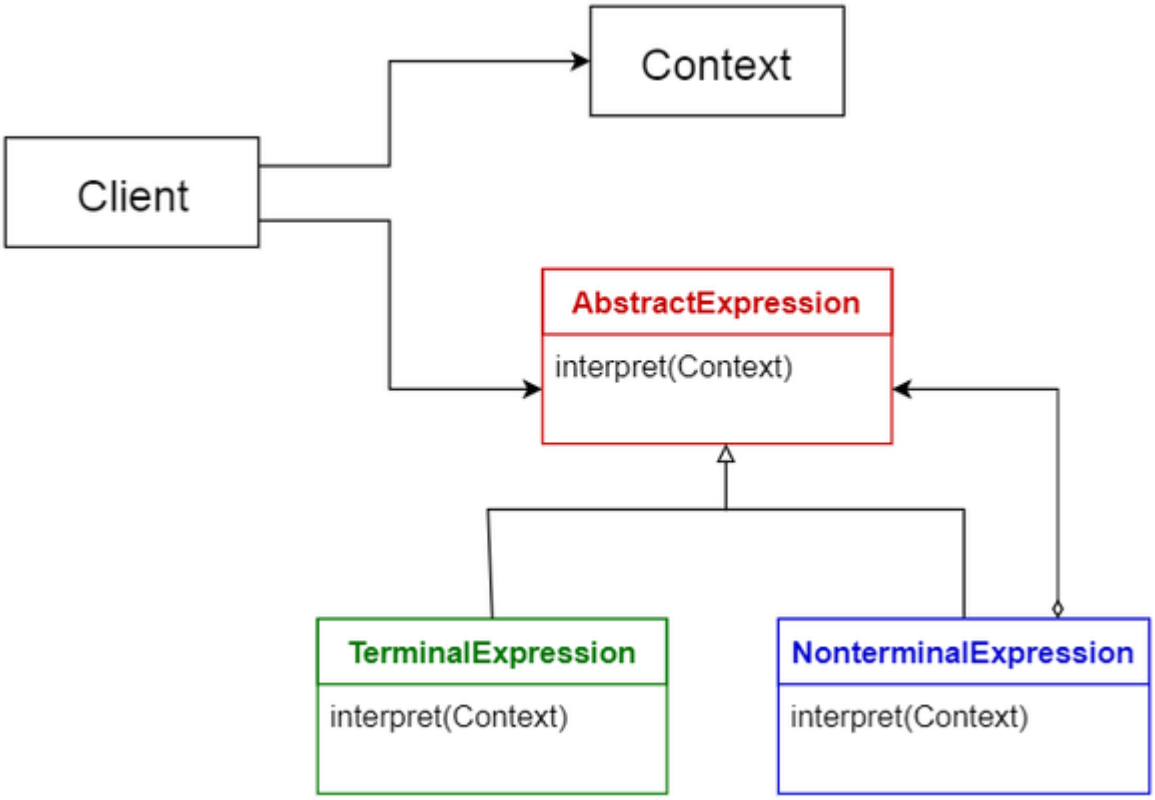
- Мови запитів до бази даних (SQL);
- Мови, які використовуються для опису протоколів передачі даних.

Плюси:

- Можливо високо динамічною поведінку.
- Відмінне рішення для кінцевих користувачів з точки зору зручності програмування.

Мінуси:

- Проблеми з продуктивністю, тому що інтерпретований мову повільніше скомпільованої.



2.1.2. Необхідність Node.js для архітектурного шаблону

Front-end розробники мають досить довгу і складну історію в розробці програмного забезпечення. Довгий час, матеріал, який надсилався до браузера, був "досить легким" і це міг зробити кожен. Тому не було спеціальної потреби в їх спеціалізації. Багато хто стверджував, що так звані веб-розробники - це не що інше, як графічні дизайнери, що використовують додаткові технології.

JavaScript - це технологія, яка справді почала змінювати сприйняття веб-розробників, перетворюючи їх на інженерів. Ця с першого погляду, маленька іграшкова мова, на яку багато інженерів-програмістів відвертала ніс, стала рушійною силою Інтернету. CSS та HTML почали активно використовуватися, оскільки було представлено більше браузерів, що створило несумісність між різними веб-браузерами, що дуже чітко визначало

потребу кваліфікованих інженерів. Тому сьогодні front-end розробники - одні з найбільш затребуваних кандидатів.

Навіть після буму Ajax, front-end інженер розглядався як головний користувач технологіями в середині вікна браузера. HTML, CSS та JavaScript були основними пріоритетами, і вони надалі б вивчали back-end-сервера (веб-сервера) тільки для того, щоб переконатися, що він належним чином виводить front-end. У певному сенсі було два шари інтерфейсу: той , що у самому браузері та той, що на сервері, який генерував дані для браузера. Розробники мали недостатній контроль над back-end інтерфейсів і часто дотримувалися думок інженерів про те, як слід складати і використовувати фреймворки разом,
які рідко враховували потреби front-end інтерфейсу.

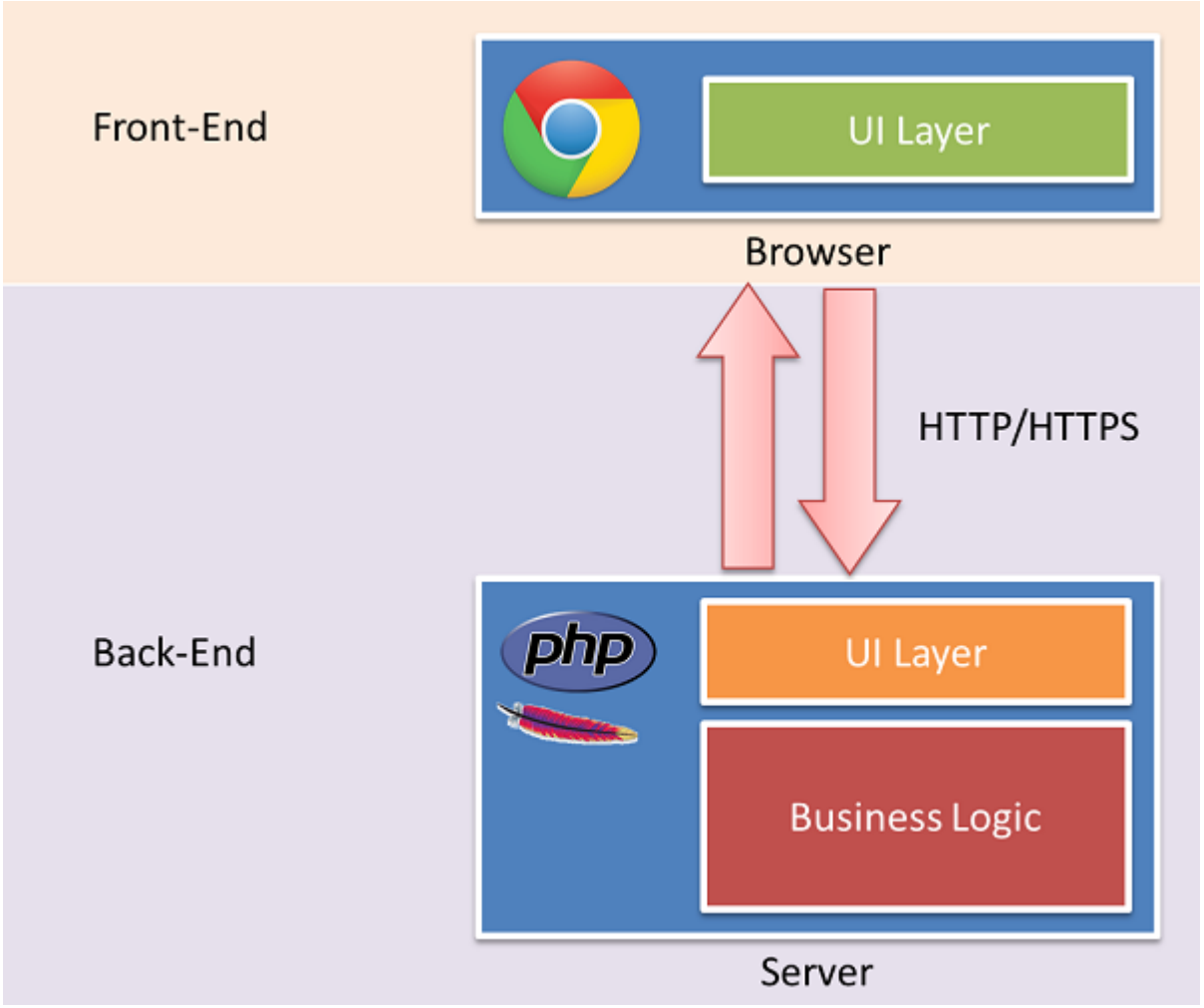


Рис. 5 Типова архітектура "Клієнт-сервер"

У цій архітектурі веб-додатків шар користувацького інтерфейсу в браузері був єдиною областю front-end розробників. Back-end інтерфейс – це місце, де зустрілися інженери обох категорій, а тоді решта архітектури сервера була там, де знаходиться сама програма. Там розташовувалися технології обробки даних, кешування, автентифікації та всі інші фрагменти функціональності, які були важливими для програми. У певному сенсі, back-end інтерфейс (часто у вигляді шаблонів) являв собою тонкий шар всередині сервера, який обслуговував лише front-end.

Таким чином, front-end - це браузер, а все інше - back-end. І так було до недавнього часу.

Коли Node.js був вперше випущений, він запалив ентузіазм серед інженерів, рівня якого не було видно з моменту введення терміна "Ajax". Ідея писати JavaScript на сервері - це місце, куди ми йдемо лише при великій необхідності - була надзвичайно вільною. Більше не потрібно блукати в PHP, Ruby, Java, Scala або іншій серверній мові. Як це доводилося робити спочатку. Якщо сервер міг бути написаний на JavaScript, то використання мов програмування для створення web-додатку лімітувалося: HTML, CSS та JavaScript.

Заміна на Node.js всього, що використовується на сервері, не завжди досить гарна ідея. Те, що він здатний це зробити, дивує та розширює можливості, але це не робить його необхідним у будь-якій ситуації.

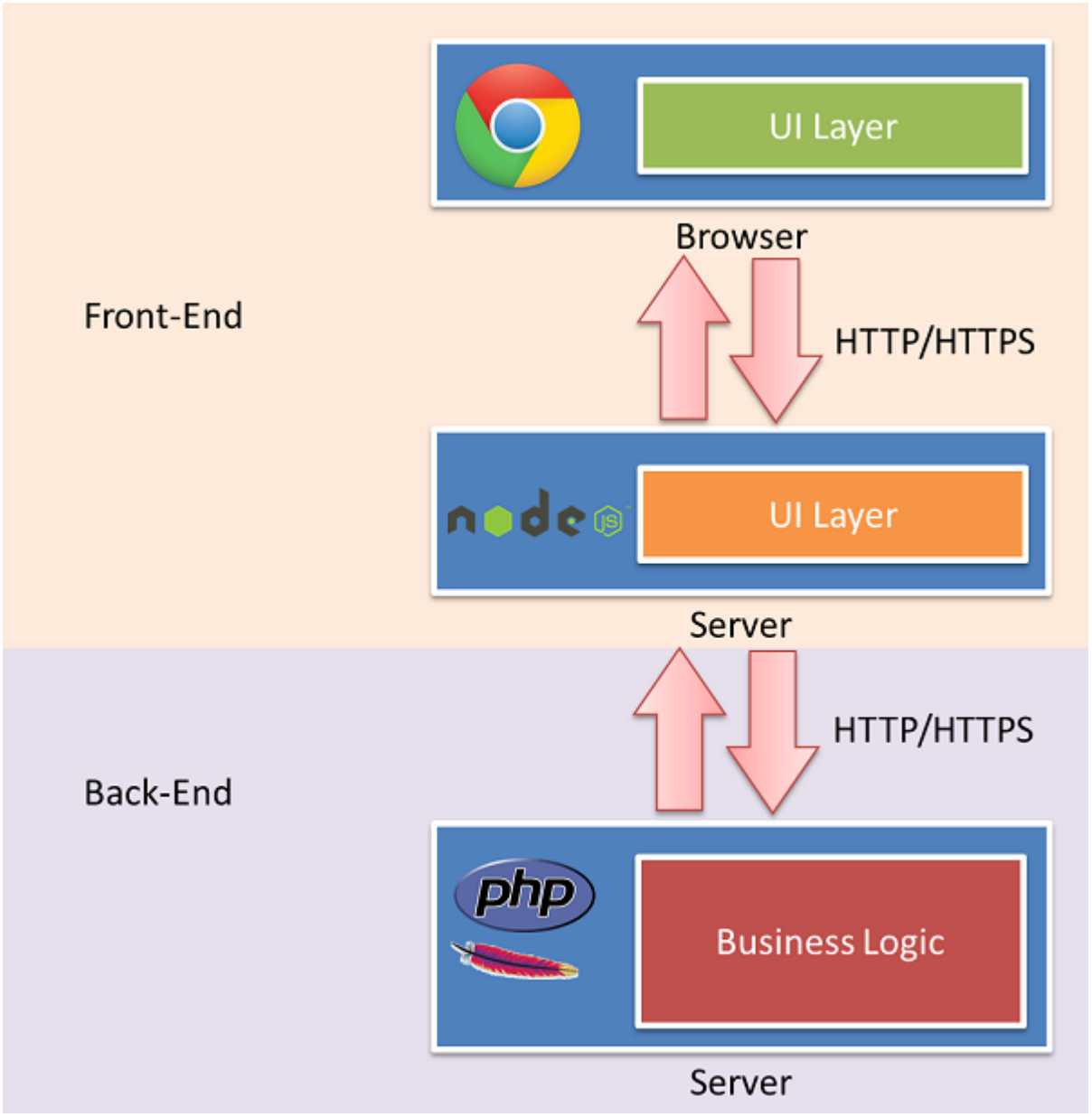


Рис. 6 Архітектура "Клієнт-сервер" з використанням Node.js

З багатьма компаніями, що рухаються до архітектури, орієнтованої на сервіс та інтерфейси RESTful, тепер стає можливим розділити рівень back-end на свій власний сервер. Якщо вся ключова бізнес-логіка програми застосована в REST-викликах, тоді потрібна лише можливість робити REST-виклики для створення цієї програми. Виникає питання. Чи back-end інженери хвилюються як відвідувачі переходять зі сторінки на сторінку? Чи їх хвилює, як здійснюється перехід, за допомогою Ajax або з оновленням повної сторінки? Чи їм все одно, використовуєте ви jQuery чи YUI? Взагалі,

аніскільки. Їм важливо щоб дані зберігалися, отримувалися та оброблялися безпечним і послідовним способом.

І саме тут Node.js надає front-end інженерам багато сили. Back-end інженери можуть писати свої REST виклики будь-якою мовою. Front-end інженери, можуть використовувати Node.js для створення back-end інтерфейсу з використанням чистого JavaScript. І це можливо виконуючи REST виклики. Front-end і back-end тепер мають ідеальний розділ проблем серед інженерів, які працюють над цими частинами. Front-end розширився до серверу, де зараз існує Node.js, а решта стека залишається підконтрольною back-end інженерам.

Ці посягання front-end в те, що було традиційно back-end, лякають багатьох інженерів які надалі вважають JavaScript "іграшковою" мовою.

Але так не повинно бути. Відокремлення шару back-end інтерфейсу від бізнес-логіки має сенс у такій великій веб-архітектурі. Чому front-end інженери повинні дбати про те, яку мову на сервері необхідно використовувати для створення функціоналу? Потреби у front-end принципово відрізняються, від потреб back-end. Якщо розробник знайомий з принципами єдиної відповідальності, розділення проблем і модульності, то досить не розумно, що раніше вони не мали цього розмежування.

До появи Node.js. не було гідного варіанту для front-end інженерів, щоб самостійно будувати back-end інтерфейс.

Все-таки весь back-end не слід писати тільки на Node.js, хоча він має таку можливість. Node.js надає розробникам можливість повністю контролювати back-end, що дозволяє виконувати роботу більш ефективно. Вони найкраще знають, як створити якісний інтерфейс зі сторони користувача та мало піклуються про те, як back-end займається обробкою даних. Потрібно було лише реалізувати можливість зрозумілого отримання необхідних даних та вказувати діловій логіці, що робити з даними. І внаслідок цього з’явилася можливість створювати гарні, якісні та доступні інтерфейси, які подобаються користувачам.

2.2. Проектування веб-додатку

2.2.1. Постановка завдання

Згідно поставленої задачі теми диплома, необхідно створити веб додаток який надає можливість планувати персональних задач та слідкувати за їх кінцевим терміном виконання. Для цього необхідно підтримувати можливість додавати, редагувати, групувати та видаляти завдання. Одні з головних завдань - це можливість відмічати виконані задачі, видаляти та архівувати їх. Для розширення функціоналу також необхідно надати можливість встановлювати так званий «дедлайн». Весь стек програми реалізовується на серверній частині клієнта, за допомогою середовища виконання Node.js який виконаний з допомогою JavaScript.

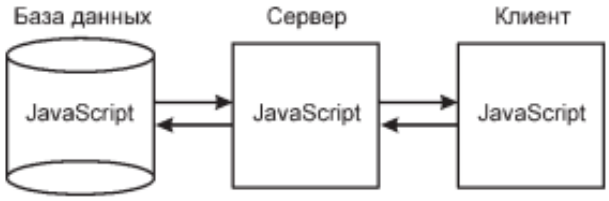
2.2.2. Огляд шаблону веб-додатка

Перед початком розробки додатку, необхідно визначитися з тим що ми розробляємо. В ході проектного аналізу було розглянуто основні типи веб-додатків які можливо створити за допомогою базових технологій, зокрема Node.js та архітектурних шаблонів які вони підтримують.

Для того, щоб майбутній додаток був сучасним та не створював складнощів при розробці. Необхідно намагатися використовувати популярні технології розробки, які існують на момент написання дипломного проекту. В ході аналізу було вирішено створити одно сторінковий додаток (Single Page Application - SPA) використовуючи архітектурний шаблон «модель-вид-контролер».

SPA – це тип програм, які працюють в браузері і не перезавантажують сторінку під час роботи. Даний тип додатків призначений для покращення та оптимізації графічного інтерфейсу користувача. Ми можемо розглядати SPA як «товстий клієнт», що завантажується з веб-сервера.

На даний момент вже існує можливість проектування та створення масштабних SPA в яких на всіх рівнях стека застосовується JavaScript.



На сьогоднішній день написані на JavaScript SPA – кращий з трьох існуючих варіантів їх побудови: Java-аплети, Flash / Flex і JavaScript. Але щоб стати конкурентоспроможним засобом створення SPA, та й взагалі претендувати на цю роль, JavaScript пройшов довгий шлях.

У міру становлення JavaScript його недоліки були в основному виправлені або пом'якшені, а переваги виступили на передній план.

Веб-браузер - найпоширеніший в світі додаток. У багатьох користувачів вікно браузера відкрито протягом усього робочого дня. Для доступу до JavaScript-додатків досить одного клацання по закладці.

JavaScript в браузері - одна з найпоширеніших у світі середовищ виконання.

Розгортання JavaScript-додатки - тривіальна задача. Щоб зробити JavaScript-додаток доступним мільярду користувачів, досить розмістити його на HTTP-сервері.

JavaScript корисний для крос-платформної розробки. Тепер ми має можливість створювати SPA у своїй улюбленій операційній системі, будь то Windows, Mac OS X або Linux, і розгортати його не тільки на будь-якому настільному ПК, але і на смартфонах і планшетах.

JavaScript став працювати на багато швидше і в деяких випадках може змагатися навіть з компільованими мови. Таке прискорення - наслідок безперервної запеклої конкуренції між Mozilla Firefox, Google Chrome, Opera і Microsoft IE. В сучасних реалізаціях JavaScript застосовуються такі передові

методи оптимізації, як JIT-компіляція в машинний код, пророкування розгалужень, виведення типу і багато поточності.

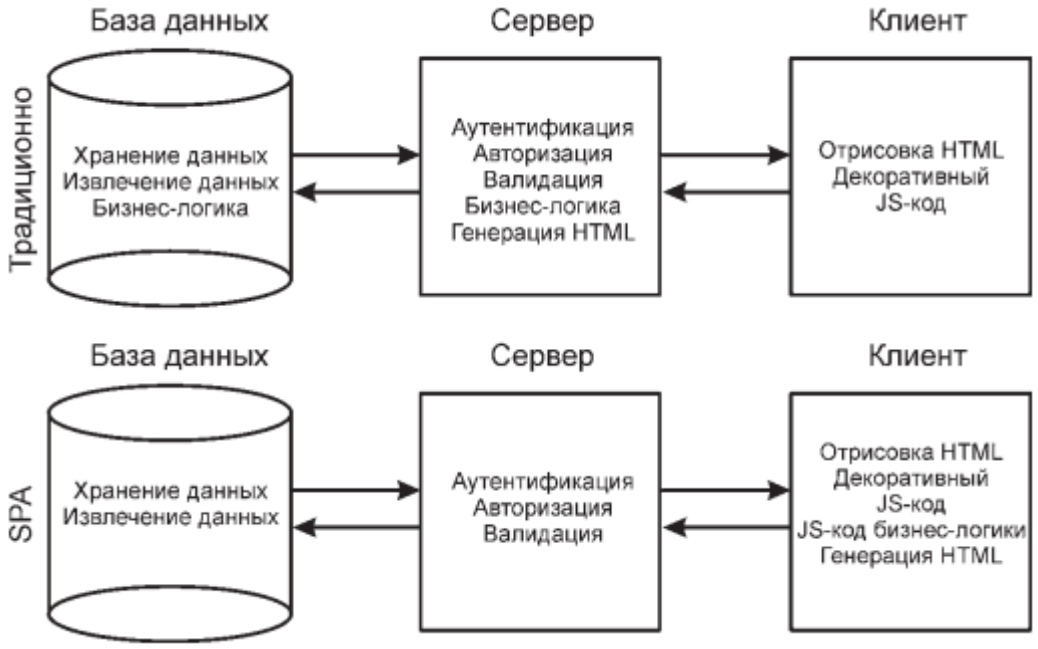
У мову JavaScript включені додаткові можливості. До них відносяться вбудований об'єкт JSON, вбудовані селектори за зразком jQuery і більш послідовно реалізовані засоби AJAX.

Стандарти HTML5, SVG та CSS3 і їх підтримка в браузерах продовжують розвиватися. Завдяки такому розвитку відображення графіки з по піксельною достовірністю може по швидкості і якості змагатися з Java і Flash.

JavaScript має можливість використовувати на всіх рівнях проекту веб-додатків. Тепер існує чудовий веб-сервер Node.js та різні типи сховища даних, які вміють передавати і зберігати дані в форматі JSON, «рідному» для JavaScript. З’явилася можливість використовувати на стороні браузера і сервера одні і ті ж бібліотеки. Настільні ПК, ноутбуки і навіть мобільні пристрої стають все більш потужними. Повсюдна наявність багатоядерних процесорів і гігабайтів пам'яті означає, що обробку, яка раніше вироблялася на сервері, можна перенести на сторону клієнта в браузері.

Завдяки зазначеним перевагам, написані на JavaScript SPA стають дедалі популярнішими, а отже, зростає попит на досвідчених розробників і архітекторів, які впевнено володіють технологіями для JavaScript.

У SPA можуть використовуватися будь-які серверні технології. Оскільки значна частина веб-додатка переміщається в браузері, то вимоги до сервера можна істотно послабити. На рис.# показано, як бізнес-логіка і генерація HTML-коду переносяться з сервера на клієнт.



Щодо архітектурного шаблону, для кращого структурування файлової системи додатку, доречно використати типове рішення MVC.

Дане рішення дозволяє розділити систему на три окремі частини:

- **Модель** – в даному випадку, це дані які надходять з серверної частини. Будь-який інтерфейс отриманий з серверної частини , буде отриманий із моделі.
- **Представлення** – це користувацький інтерфейс, який користувач має можливість бачити та взаємодіяти. В даному випадку, він буде генеруватися динамічно на основі поточної моделі додатку.
- **Контролер** – знаходиться на рівні бізнес-логіки та представлення, котрий виконує службові функції.

2.2.3. Структура додатку

Даний додаток має модульну структуру, де кожен модуль є самостійною і незалежною одиницею. Структура каталогів додатку:

Assets/icons – каталог для зберігання іконок інтерфейсу

config – містить файл конфігурації сервера

nitro.sdk – зберігаються готові службові сценарії подій, взаємодій з базою даних та іншої реалізації внутрішнього функціоналу додатку.

nitro.ui – основний каталог графічного інтерфейсу з розташованим *index.html*, сценаріями відображення стилів та головним index.js.

node_modules – каталог для зберігання необхідних модулів.

build-scripts/inject-scripts.js – зберігається сценарій для побудови кінцевого index.html який буде виконуватися на сервері.

До компоненти клієнтського інтерфейсу належать всі каталоги, крім node_modules.

Тільки каталог node_modules відноситься до серверної частини інтерфейсу, так як вся реалізація

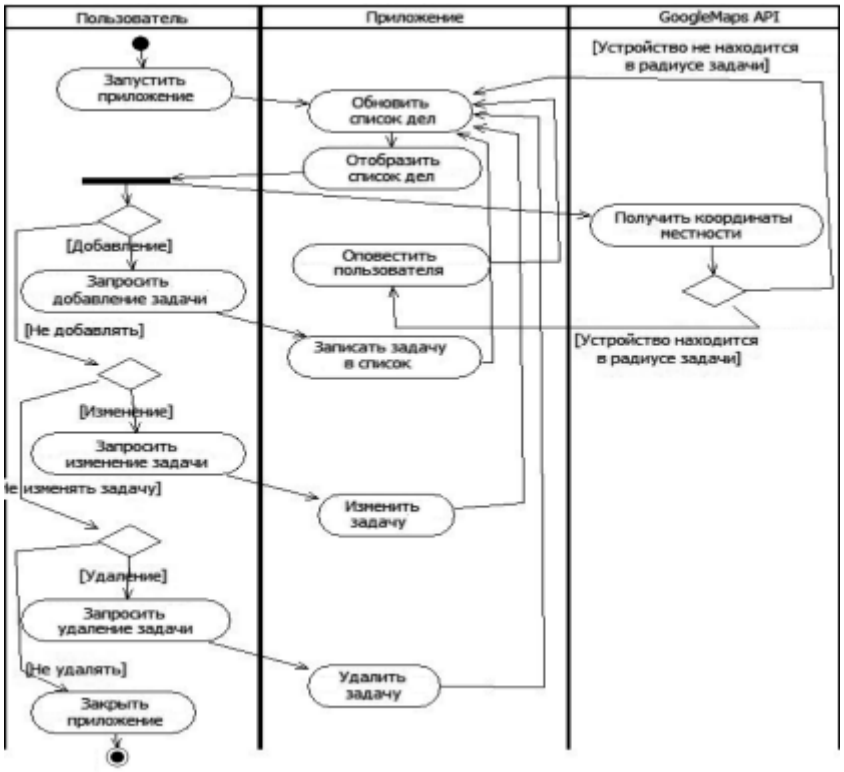
2.2.4. Формалізація вимог додатка

Для формалізації вимог додатка була розроблена діаграма варіантів використання(Use Case) Рис.# для визначення функціональних вимог майбутнього додатка. Дана діаграма варіантів використання розмежовує додаток і його оточення та так само відображає, як користувач взаємодіє з додатком. У користувача є можливість переглядати поточний список справ і кожну задачу окремо, створювати персональні списки, переглядати задачі в хронологічній послідовності. При додаванні завдання, користувач вводить такі параметри завдання: назва завдання, примітка до задачі, дата виконання та кінцевий термін для виконання. Також додаток надає можливість змінювати та видаляти кожну задачу.

Представлена діаграма відображає необхідний функціонал веб-додатку та через які варіанти використання виконується взаємодія користувача з системою.

2.2.5. Логіка роботи додатку

Використовуючи визначені функціональні вимоги створюваного додатку, на даному етапи необхідно побудувати модель логіки роботи програми. Для вирішення даної задачі була розроблена діаграма активності Рис.#. Дана діаграма відображає взаємодію користувача та додатка. Необхідно відмітити, що на головному екрані додатка відображаються не груповані задачі. По завершенню кожної дії користувача з додатком, ініціюється з сервером процеси оновлення та відображення змін.



2.2.6. Результат

В ході виконання даного моделювання був спроектований повний та необхідний функціонал додатку та графічний інтерфейс користувача, а саме меню для відображення створених та постійних списків, та основної частини додатку. Яке включає поле для перегляду та створення нових завдань.

ВИСНОВОК ДО РОЗДІЛУ 2

В даному розділі було проаналізована існуючі архітектурні шаблони для веб-додатків та охарактеризована доцільність використання Node.js для реалізації обраного архітектурного шаблону програмного забезпечення. Найбільш прийнятним на даний момент, виявилась розробка одно сторінкового додатку. На даний момент SPA – це не нововведення в мережі інтернет, адже вони існують доволі довгий час. Та в SPA в якості веб-додатку тільки набирає обертів, через відсутність в минулому доступних технологій, які б надавали прості механізми створення даних додатків. Після детального аналізу було спроектовано весь необхідний функціонал додатку , який відповідає сучасним критеріям розробки графічних інтерфейсів користувача. Та змодельовані процеси які виникають від час взаємодії користувача з додатком та додатка з сервером.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1. Основні модулі npm для Node.js

Модулі і модульна архітектура - це основа Node.js. Модулі забезпечують інкапсуляцію (приховуючи подробиці реалізації та відкриваючи тільки інтерфейс за допомогою module.exports), повторне використання коду, логічне розбиття коду на файли. Практично всі програми Node.js складаються з безлічі модулів, які повинні якимось чином взаємодіяти. Якщо неправильно пов'язувати модулі або взагалі не слідкувати за взаємодією модулів, то можна дуже швидко виявити, що додаток починає «розвалюватися»: зміни коду в одному місці призводять до поломки в іншому, а модульне тестування стає просто неможливим.

NPM - найбільший у світі реєстр програмного забезпечення. Код якого є відкритим, на всіх континентах використовують npm для обміну та запозичення пакетів, а багато організацій використовують npm і для управління приватною розробкою.

NPM складається з трьох різних компонентів:

- веб-сайт
- інтерфейс командного рядка (CLI)
- реєстр

Використовуючи веб-сайт, наявна можливість відкрити пакунки, налаштувати профілі та керувати іншими аспектами свого npm-досвіду. Наприклад, можливість налаштувати Orgs (організації) для управління доступом до державних або приватних пакетів. CLI працює від терміналу, так, як більшість розробників взаємодіють з НПМ завдяки командному рядку. Реєстр є великою громадською базою даних програмного забезпечення JavaScript і мета-інформації навколо нього.

NPM використовують для:

- Адаптації пакети коду своїх додатків або використання таких пакетів, якими вони є.
- Завантаження автономних інструментів, якими моєливо скористатися відразу.
- Запуску пакетів без завантаження за допомогою npx .
- Поширення коду з будь-яким користувачем npm, в будь-якому місці.
- Обмеження коду конкретними розробниками.
- Створення віртуальних команд за допомогою Orgs.
- Управління кількома версіями коду та кодової залежності.
- Простого оновлення програми, коли основний код оновлюється.
- Пошуку інших розробників, які працюють над подібними проблемами та проектами.

В каталозі node_modules знаходяться всі модулі, з яких складається додаток Node.js. У ньому за умовчанням встановлюються модулі npm, які визначено в package.json. Модулі які створюються власноруч також повинні розташовуватися в каталозі.

Замість прямої вказівки модуля в командному рядку, наприклад React, ми скористаємося файлом маніфесту, який називається package.json, щоб повідомити npm, які версії та яких модулів необхідні встановити для нашого додатку, щоб він працював . Це зручно, коли ми хочемо встановити додаток на віддалений сервер або коли хтось інший викачує і встановлює наш додаток на свою машину.

3.1.1. Webpack

На даний момент, модулі в браузерах рідше коли використовуються в «сирому» вигляді. Зазвичай їх об'єднують разом, використовуючи спеціальний інструмент - Webpack і після цього тільки викладають код на робочий сервер.

Одна з переваг використання збирача - він надає більший контроль над тим, як здійснюється пошук модулів в самій програмі, що дозволяє використовувати «голі» модулі та багато іншого, наприклад CSS / HTML-модулі.

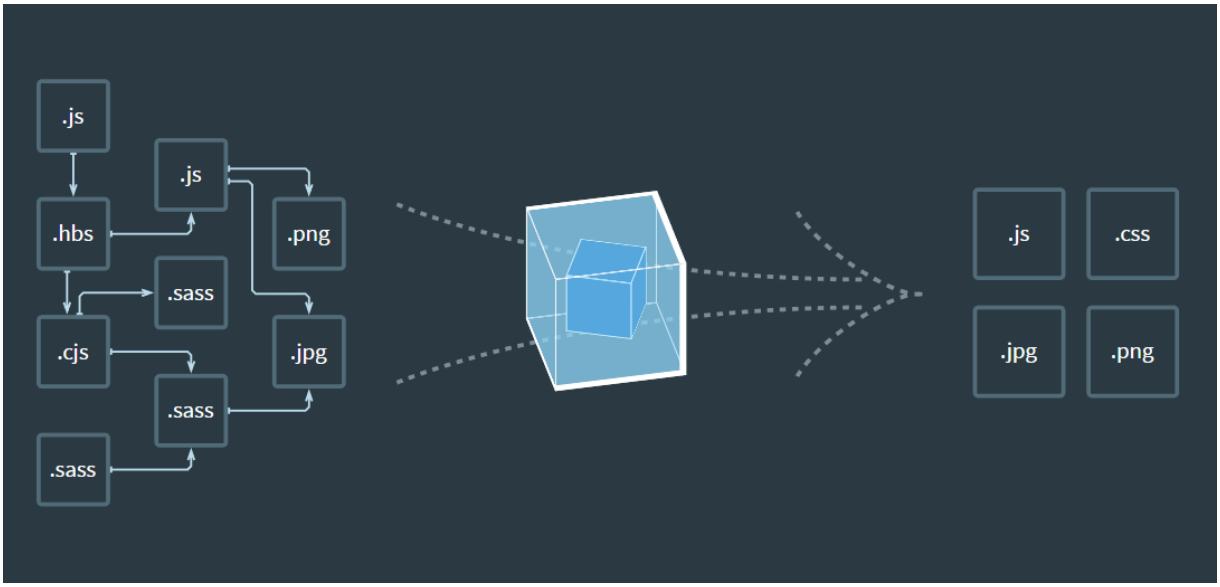
Під час запуску Webpack, він робить наступне:

1. Бере «основний» модуль, який ми збираємося помістити в HTML, використовуючи дану конструкція:

   <script type = "module">

2. Аналізує залежності (import, імпорти імпортів і так далі).
3. Збирає один файл з усіма модулями (або кілька файлів, це можна налаштувати), перезаписує вбудований import функцією імпорту від збирача, щоб все працювало. «Спеціальні» типи модулів, такі як HTML та CSS теж підтримуються.
4. В процесі можуть відбуватися й інші трансформації та оптимізації коду:
   - Недосяжний код видаляється.
   - Невикористані експорти видаляються («tree-shaking»).
   - Специфічні оператори для розробки, такі як console та debugger, також видаляються.

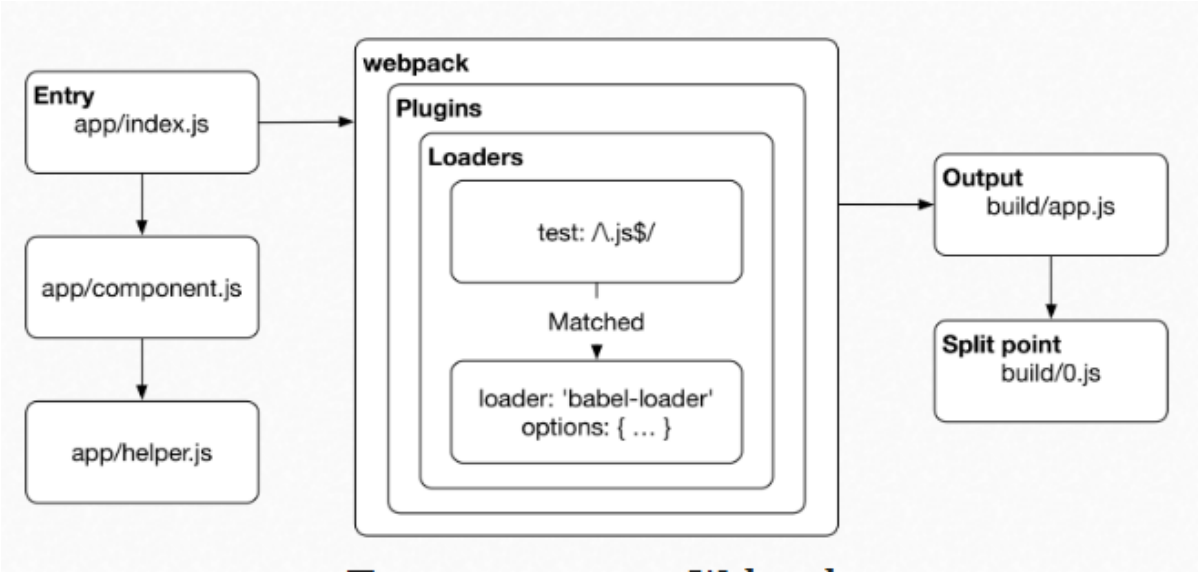
Та вже отриманий файл можна мінімізувати (видалити пробіли, замінити назви змінних на більш короткі і т.д.).



Найменший проект, який можливо поєднати з веб-пакетом, складається з введення та виведення . Процес поєднання починається з визначених користувачем записів . Самі записи є модулями і можуть вказувати на інші модулі через імпорт .

Коли необхідно зв'язати проект за допомогою webpack, він обходить імпорт, будуючи графік залежності проекту, а потім генерує вихід на основі конфігурації. Крім того, можна визначити розділені точки для створення окремих пакетів у самому коді проекту.

Вебпак підтримує формати модулів ES2015, CommonJS та AMD поза коробкою. Механізм навантажувача працює і для CSS, @import та url() підтримує через css-loader. Також існують плагіни для конкретних завдань, таких як мінімізація, інтернаціоналізація, HMR тощо.



В даному випадку ми використовуємо Webpack 4 який надає можливість розділяти зібраний файл(бандл) на 3 частини.

3.1.2. React

Як ми вже знаємо, React - це бібліотека JavaScript для створення інтерфейсів користувача. Даний пакет містить тільки функціональні можливості, необхідні для визначення компонентів. Зазвичай він використовується разом з рендерінгом, як react-dom для Інтернету або react-native для рідного середовища. За замовчуванням React знаходиться в режимі розробки. Версія для розробки включає додаткові попередження про поширені помилки, тоді як виробнича версія включає додаткові оптимізації продуктивності та знімає всі повідомлення про помилки.

3.1.3. React-dom

Даний пакет служить точкою входу для DOM та сервером рендерів для React. Він призначений для парного з'єднання з загальним пакетом React

3.1.4. React-native-web

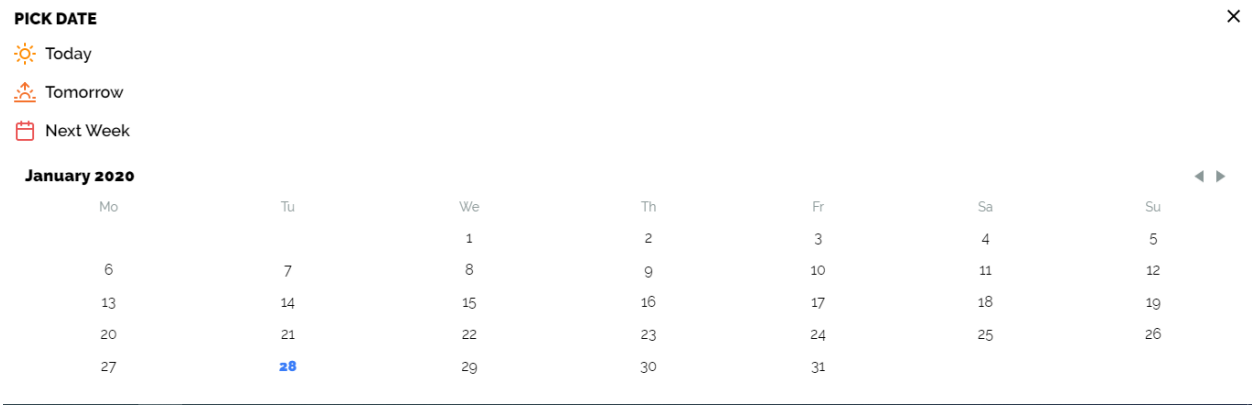
React Native for Web дає можливість запускати компоненти та API React Native в Інтернеті за допомогою React DOM.

Якісний веб-інтерфейс : спрощує створення швидких , адаптивних веб-інтерфейсів у JavaScript. Він надає взаємодію якості рідної якості, підтримку декількох режимів введення (сенсорний, мішаний, клавіатурний), оптимізованих стилів, встановлення для постачальників, вбудовану підтримку верстки RTL, вбудовану доступність та інтеграцію з інструментами React Dev.

Можливість розробити один раз, візуалізувати де завгодно : взаємодіє з існуючими компонентами React DOM і сумісний з більшістю API React Native. Наявна можливість розробити нові компоненти для рідної та веб-сторінки, не переписуючи існуючий код. React Native for Web також може надавати HTML та критичний CSS на сервері за допомогою Node.js.

3.1.5. React-day-picker

Гнучка складова вибору дати для React. Високо налаштовується,



локалізується, з підтримкою ARIA, без зовнішніх залежностей.

3.1.6. React-router, React-router-dom, React-router-native

Модуль для декларативної маршрутизації React. Даний пакет забезпечує основну функціональність маршрутизації для React Router, однак, іноді його не потрібно встановлювати безпосередньо. Якщо створюється програма, яка працюватиме в браузері, замість цього слід встановити react-router-dom. Так само, якщо створюється заявка для React Native, слід

встановити її react-router-native. Будь-який з них встановлює react-router як залежність.

3.1.7. History

Бібліотека history дозволяє легко керувати історією сеансів у будь-якому місці роботи JavaScript. Вона абстрагує відмінності в різних середовищах і надає мінімальний API, який дозволяє керувати стеком історії, навігацією та збереженням стану між сесіями.

Бібліотека історії - це легкий шар над вбудованими API історії та місцеположення браузера. Його мета полягає не в тому, щоб забезпечити повну реалізацію цих API, а в тому, щоб полегшити користувачам доступ до різних методів навігації.

3.1.4. Babel

Babel - це транспілер, який в основному використовується для перетворення коду ECMAScript 2015+ у зворотну сумісну версію JavaScript у поточних та старих браузерах чи середовищах. Ось основні речі, які Babel може зробити :

- Перетворити синтаксис
- Функції Polyfill, які відсутні у нашому цільовому середовищі (через @Babel / polyfill)
- Перетворення вихідного коду (кодемоди)

Транспілятори або компілятори від джерела до джерела - це інструменти, які читають вихідний код, написаний однією мовою програмування, і виробляють еквівалентний код іншою мовою, що знаходиться на тому ж рівні.

Трансліпінг - це процес взяття однієї мови та перетворення її на іншу, наприклад перетворення TypeScript с у javascript та sass у css.

Компіляція перетворює одну мову в іншу на нижчому рівні абстракції, наприклад, Java в байт-код, а трансліпування конвертує одну мову в іншу на тому ж рівні абстракції, як typecript в javascript або sass в css.

В основному використовуємо через те, що всі останні функції JavaScript ще не підтримуються у кожному браузері. Тому babel використовують, щоб перекласти останні функції javascript (функції ES6), які не зрозумілі кожному браузеру в ES5, зрозумілі для кожного браузера.

3.1.5. Файли маніфесту

Package.json

Package.json файл є основним для екосистеми Node.js і є базовою частиною розуміння та роботи з Node.js, npm та навіть сучасним JavaScript. Він прирівнюється до маніфесту про додатки, модулі, пакети і багато іншого - це інструмент , який використовується , щоб зробити сучасні розробки спрощеними, модульними і ефективними.

Усі пакети npm містять файл, як правило, в корені проекту, який називається package.json- цей файл містить різні метадані, що стосуються проекту. Цей файл використовується для надання інформації, npm яка дозволяє йому ідентифікувати проект, а також обробляти залежності проекту. Він також може містити інші метадані, такі як опис проекту, версія проекту в певному розповсюдженні, інформація про ліцензії, навіть дані про конфігурацію - все це може бути життєво важливим як npmдля кінцевих користувачів пакету. package.json файл зазвичай знаходиться в кореневому каталозі проекту Node.js.

Інформацію про метадані у файлі package.json можна класифікувати за наступними категоріями:

1. Визначення властивостей метаданих: восновному вона складається з властивостей для ідентифікації модуля / проекту, таких як назва проекту, поточна версія модуля, ліцензія, автор проекту, опис проекту тощо.

2. Функціональні властивості метаданих: Як випливає з назви, він складається з функціональних значень / властивостей проекту / модуля, таких як вхід / початкова точка модуля, залежності проекту, сценарії використовуються, посилання на сховища проекту Node тощо.

Щоб вказати пакети, від яких залежить проект, необхідно перелічити їх як "dependencies" або "devDependencies" у package.json файлі пакету . Коли розробник (або інший користувач) запустить npm install, npm завантажить залежності та devDependpendents, які перераховані в списку, package.json, що відповідають вимогам семантичної версії, переліченим для кожного.

"dependencies": Пакети, необхідні вашою заявкою у виробництві.

"devDependencies": Пакети, необхідні лише для місцевого розвитку та тестування.

Package-lock.json

Package-lock.json автоматично генерується для будь-яких операцій, де npm модифікує або node_modulesдерево, або package.json. Він описує точне дерево, яке було сформовано таким чином, що наступні установки можуть генерувати однакові дерева, незалежно від проміжних оновлень залежності.

Цей файл призначений для введення в сховища джерел і служить різним цілям:

- Опишіть єдине представлення дерева залежностей таким чином, що колеги, розгортання та безперервна інтеграція гарантовано встановлюють однакові залежності.
- Забезпечте можливість для користувачів "подорожувати часом" до попередніх станів, node_modulesне потребуючи привласнення самого каталогу.
- Для полегшення більшої видимості змін дерев через читабельний контроль джерела відрізняється.

- І оптимізуйте процес встановлення, дозволяючи npm пропускати повторні розділення метаданих попередньо встановлених пакетів.

Однією з ключових деталей package-lock.jsonє те, що вона не може бути опублікована, і вона буде ігнорована, якщо знайдеться в будь-якому іншому місці, крім пакету топлерів.

Webpack.config.js

Без пакування веб-пакет не вимагатиме використання конфігураційного файла. Однак буде припускати, що вхідним пунктом вашого проекту є src/indexі отримає результат в dist/main.js мінімізованому вигляді та оптимізованому для виробництва.

Зазвичай ваші проекти потребують розширення цієї функціональності, для цього ви можете створити webpack.config.jsфайл у кореневій папці, і вебпакет автоматично використовуватиме його.

Файл конфігурації webpack - це файл JavaScript, який експортує конфігурацію веб-пакету . Потім ця конфігурація обробляється веб-пакетом на основі визначених властивостей.

Оскільки це стандартний модуль Node.js CommonJS, має можливість зробити наступне :

- імпортувати інші файли через require(...)
- використовувати утиліти на npm via require(...)
- використовувати вирази потоку керування JavaScript, наприклад ?:оператор
- використовувати константи або змінні для часто використовуваних значень
- записувати та виконувати функції для створення частини конфігурації
  var path = require('path');

```
module.exports = {
  mode: 'development',
  entry: './foo.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'foo.bundle.js'
  }};
```

3.2. Внутрішня реалізація додатка

3.2.1. Файлова та головна реалізація основних персональних модулів

До основних файлів створеного додатку належать:

- index.html
- index.js
- styles.js
- app.jsx
- shell/index.jsx
- root.css
- sdk/index.js
- assets.json

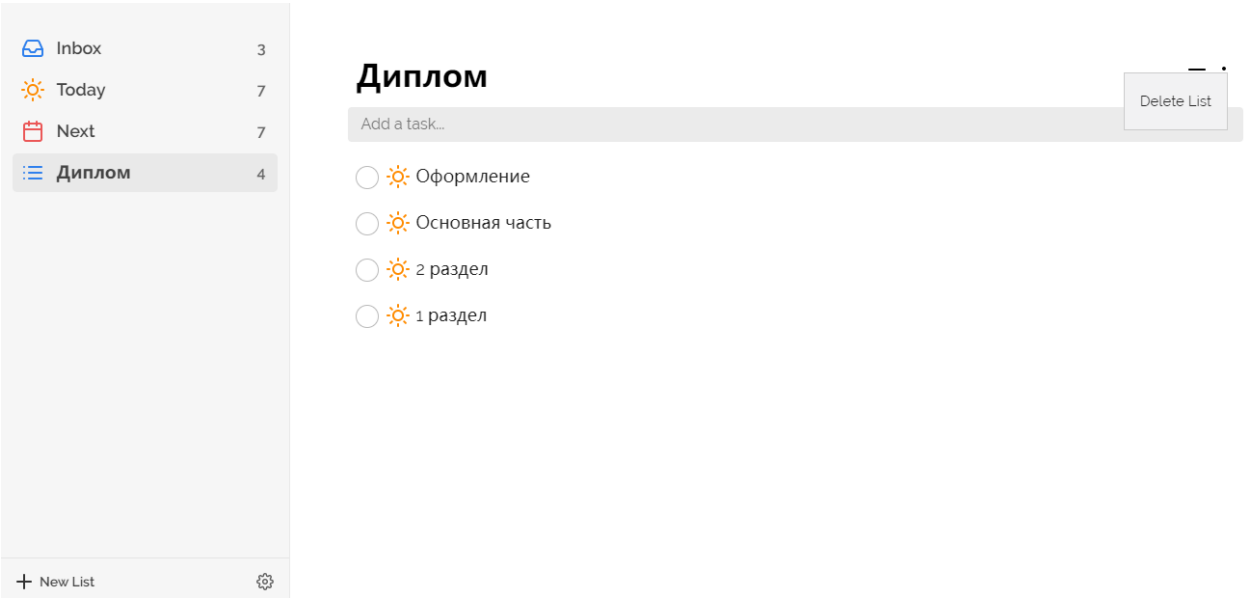
index.html **–** це і є SPA, який ми передаємо на локальний сервер для відображення. Зміст цього файлу постійно динамічно генерується , за допомогою модулів Node.js. Статичний код який в ньому зберігається має вигляд:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```
  <title><%= htmlWebpackPlugin.options.title %></title>
  <link rel="icon" type="image/png" href="/img/favicon/favicon-48x48.png" sizes="48x48" />
  <link rel="icon" type="image/png" href="/img/favicon/favicon-32x32.png" sizes="32x32" />
  <link rel="icon" type="image/png" href="/img/favicon/favicon-16x16.png" sizes="16x16" />
  <link rel="mask-icon" href="/img/icons/safari-pinned-tab.svg" color="#3a7df8">
  <meta name="theme-color" content="#3a7df8" id="theme">
  <link rel="manifest" href="/manifest.json">
  <style>
    body {
      margin: 0;
    }
    #loading-wrapper {
      text-align: center;
      font-family: BlinkMacSystemFont,"Segoe UI",Roboto,Oxygen-Sans,Ubuntu,Cantarell,"Helvetica Neue",sans-serif;
      padding-top: 2rem;
      color: #666;
    }
    #loading-wrapper p { font-weight: 600; }
    #loading-button { font-size: 1rem; }
  </style>
</head>
<body>
  <div id="app-shell">
```

```
    <div id="loading-wrapper">
        <p id="loading">Loading...</p>
    </div>
    <script>
    </script>
  </div>
</body>
</html>
```

Та вже після запуску серверу та введення http://localhost:8080 в адресну



строку браузера. Ми отримуємо такий результат:

index.js – це основний файл в якому ми ініціюємо основні необхідні модулі, файли та каталоги.

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './components/app.jsx'
import { NitroSdk } from '../nitro.sdk'
```

```
// css
import './external-css/fonts.css'
import './external-css/daypicker.css'
import './external-css/extras.css'
import './external-css/root.css'
```

styles.js – файл для зберігання основних стилів відображення нашого додатку.

```
const vars = {
  // colors
  backColor: '#f2f2f3',
  boxColor: '#6eaf01',
  boxColorDark: '#629d00',
  navTextColor: '#444',
  navTextColorMuted: '#666',
  accentColor: '#3a7df8',
  accentColorMuted: '#5a92fa',
  taskTextColor: '#222',
  taskSubtextColor: '#666',
  dragColor: 'rgba(58, 125, 248, 0.1)',
  indicatorColor: '#eeeeee',
  headerColor: '#000',
  overlayColor: '#fafbfc',
  positiveColor: '#2ecc71',
  negativeColor: '#e74c3c',
  cancelColor: '#5a92fa',
  cancelBorderColor: '#ccc',
  focusBorder: '#9dbefb',
  // sizes
```

```
  padding: 16,
  taskHeight: 56,
  materialHeaderHeight: 56,
  notesLineHeight: 24,
  // font sizes
  modalFontSize: 17,
  taskFontSize: 18,
  taskFontSizeSmall: 13,
  taskExpandedFontSize: 18,
  taskHeaderFontSize: 20,
  taskInputFontSize: 16,
  // font family
  fontFamily: '"Raleway", "Microsoft YaHei", "sans-serif"'
}
export { vars }
```

app.jsx – це файл написаний на React, який реалізовує основні функції додатку. Надалі він імпортується до index.js. Для цього він імпортує такі модулі:

```
import React from 'react'
import { Router } from 'react-router-dom'
import { DragDropContext } from 'react-beautiful-dnd'
import createBrowserHistory from 'history/createBrowserHistory'
const history = createBrowserHistory()
import { NitroSdk } from '../../nitro.sdk'
import { UiService } from '../services/uiService.js'
import { Shell } from './shell/index.jsx'
import { ContextMenu } from './contextMenu.jsx'
import { Modal } from './modal.jsx'
```

shell.jsx – react модуль для реалізації оболонки компонентів додатка.

```
class ShellComponent extends React.Component {
    static propTypes = {...}
    state = {...}
    constructor(props) {...}
    componentDidMount() {...}
    handleNewCardPosition = (position, animate, manual) => {...}
    universalError = err => {...}
    togglePin = () => {...}
    toggleStations = (newPosition = 'toggle') => {...}
    getSnapAnchor(pos) {...}
    getFlickAnchor(pos) {...}
    triggerTouchStart = e => {...}
    triggerTouchMove = e => {...}
    triggerTouchEnd = e => {...}
    triggerPaddingButton(e) {...}
    triggerRootContainer = e => {...}
    render() {...}
}
const Shell = withRouter(ShellComponent)
export { Shell }
```

root.css – містить таблицю стилів з правилами для коректного відображення компонентів додатку.

sdk/index.js – сценарій який виконує службову роль, в якому реалізується службові процеси всередині додатка. Для цього він імпортує деякі модулі з каталогу sdk та описує їх реалізацію.

```
import Events from './events.js'
import { ListsCollection } from './collections/listsCollection.js'
```

```
import { TasksCollection } from './collections/tasksCollection.js'
import SyncQueue from './sync/syncQueue.js'
import SyncGet from './sync/syncGet.js'
import { broadcast } from './sync/broadcastchannel.js'
import { log, warn, error, logHistory } from './helpers/logger.js'
const systemLists = ['inbox', 'today', 'next', 'all']
const fakeLists = ['today', 'next', 'all']
```

assets.json – файл який зберігається в головному каталозі, в якому вказаний шлях до всіх

3.2.2. Побудова app-shell

App-shell - це головний контролер SPA, необхідна частина нашої архітектури. Роль модуля app-shell доречно порівняти з роллю корпусу літака.

Він координує взаємодію функціональних модулів, в яких зосереджена бізнес-логіка, з універсальними браузерними інтерфейсами. Коли користувач робить щось, що створює необхідність зміни стану додатка, модуль app-shell координує зміни.

Згідно обраної архітектури (MVC), вважаємо модуль app-shell головним контролером, оскільки він координує роботу контролерів всіх підлеглих функціональних модулів. app-shell відповідає за вирішення наступних завдань: отрисовка та керування функціональними контейнерами; управління станом додатки; координація функціональних модулів.

Модуль app-shell створює і керує контейнерами, які використовують функціональні моделі.

Головна ідея проекту - відмовитися від статичного HTML і CSS і доручити React разом з Node відображення контейнерів app-shell.

Модель - це те місце, куди app-shell і всі функціональні модулі ходять за даними і бізнес-логікою. Якщо нам потрібно аутинтифікувати користувача, то ми викликаємо метод, що надається моделлю. Бажаючи отримати список завдань, ми звертаємося до моделі.

Моделі не потрібен браузер. Це означає, що модель не повинна припускати наявності об'єкта document або таких специфічних для браузера методів, як document.location. При використанні паттерна MVC вважається хорошим стилем доручати відображення представлення даних моделі модулю app-shell і функціональним модулями.

Модель не надає ніяких універсальних утиліт. Замість цього ми користуємося загальної службової бібліотекою.

Реалізація app-shell починається з файлу з файлу index.js де відбувається імпорт модуля App за адресою './components/app.jsx'

```
import App from './components/app.jsx'
```

Використовуючи подію document.addEventListener(DOMContentLoaded) спочатку відбувається завантаження html коду, а вже потім відбвається побудова app-shell. Дана подія необхідна для контролю почергового виконування сценаріїв. Ініціалізація побудови відбувається за допомогою даної частини коду:

```
const shell = document.getElementById('app-shell')

 NitroSdk.dataLoaded = NitroSdk.loadData()

   .then(() => {

    ReactDOM.render(<App />, shell)

   })
```

Виконання дії ReactDOM.render(<App />, shell), призводить до виклику модуля app.jsx який вже реалізує весь необхідний функціонал додатку.

Відзначити частину коду app.jsx в який відбувається побудова 2 основних компонентів app-shell:

```
<React.Fragment>
    <Router history={history}>
      <DragDropContext onDragEnd={this.triggerDragEnd}>
        <Shell />
      </DragDropContext>
    </Router>
    <Modal />
    <ShortcutsModal />
    <TutorialModal />
    <ContextMenu />
  </React.Fragment>
```

Основний код для реалізації обох компонентів app-shell:

```
render() {
   const rootClassName =
     'root-container ' +
     this.state.cardPosition +
     '-view' +
     (this.state.delayCard ? ' delay-transition' : '')
   return (
     <React.Fragment>
       <div
```

```
className={rootClassName}
ref={e => (this.rootcontainer = e)}
onClick={this.triggerRootContainer}>
<div className="root-map">
  <Switch>
    <Route exact path="/" render={() => <Redirect to="/inbox" />} />
    <Route
      path="/:list/:task?"
      render={routeProps => {
        TasksExpandedService.routeUpdate(routeProps)
        return <Tasks {...routeProps} />              }}              />
  </Switch>
</div>
<div
  className="root-card enable-scrolling"
  ref={e => (this.touchcard = e)}          >
  <div
    className="root-card-padding-button"
    onTouchStart={this.triggerPaddingButton}           />
  <View className="root-card-wrapper">
    <Lists />
  </View>
```

```
        </div>
      </div>
    </React.Fragment>    )
```

3.3. Особливості додатку

3.3.1. Реалізація локального сервера та бази даних

Webpack Dev Server

Webpack-dev-server дає можливість створювати локальний сервер для швидкої розробки програми. Це обумовлено можливість виконувати живе перезавантаження під налагоджування програми. В програмі реалізовано використання серверу з сценаріями NPM. Сценарії NPM package.json є зручним і корисним засобом для запуску локально встановлених бінарних файлів, не турбуючись про їх повний шлях. Після запуску екземпляра сервера він починає встановлювати з'єднання з localhost з портом 8080. Вже одразу після встановлення пакету webpack-dev-server він за замовчуванням налаштований для підтримки живого перезавантаження файлів під час редагування файлів під час роботи серверу.

IndexedDB

Для реалізації даного додатку, не використовується база даних як така. Через те, що додаток складається з однієї простої сторінки, вміст та контент якої генерується на серверній частині додатку. При цьому, на даний момент використовується лише локальний сервер. Необхідність створювати окрему базу даних та використання системи керування базою даних просто зникла. Ми маємо можливість користуватися API клієнтського сховища даних , під назвою IndexedDB.

Тобто використовується NoSQL сховище даних в форматі JSON всередині браузера. Ця можливість була впроваджена в браузерах з 2011 року.

Це сховище даних, набагато потужніша , аніж LocalStorage і тим більше cookie. Веб-сайти або веб-додатки можуть постійно зберігати і читати дані клієнтського сховища. Кожен домен має свою власну область зберігання, де доступ до даних, що зберігаються в іншому домені, не надається відповідно до принципу однакових джерел.

Дана база даних є об'єктною базою даних, в якій зазвичай зберігаються об'єкти JavaScript, з одного із записів в якості ключа, за допомогою якого наявна можливість отримати дані.

Доступ до бази даних асинхронний: клієнт робить запит і реєструє функції зворотного виклику. Як тільки стане відомий результат запиту, він буде проінформований про це і, в разі успіху будуть надані запитані дані, а в разі помилки - причина.

У базі можливі кілька типів ключів, а значення можуть бути майже будь-якими. Для надійності вона підтримує транзакції.

IndexedDB як найкраще підходить для офлайн або локальних додатків.

Якщо додаток в майбутньому буде завантажений на статичний сервер, то доволі легко буде під'єднати іншу базу даних або залишити дану.

3.3.2. Використання Webpack

Для реалізації додатку, потрібно зібрати всі модулі до купи в один файл та відправити його на сервер. Для цього ми використовуємо Webpack який надає нам цю можливість, при цьому ми створюємо новий каталог dist в якому з'являється модернізований файл index.html. Для цього ми використовуємо два компонента:

- assets.json
- build-scripts/inject-script.js

assets.json – файл маніфесту який зберігається в головному каталозі, у якому вказаний шлях до всіх необхідних файлів для створення основного index.html

inject-script.js - сценарій для побудово нового index.html. В ньому визначено які фрагменти необхідно змінити та додати, та шлях до 3 файлів які були зібрані Webpack на базі файлу маніфесту.

```
const path = require('path')

const fs = require('fs')

const assets = require('../assets.json')

const html = path.resolve(__dirname, '..', 'dist', 'index.html')

const htmlRegex = /index\.html$/i

fs.readdir(path.resolve(__dirname, '../', 'dist'), (error, files) => {
 if (error) {
  throw new Error(error)
 }
 for (let file of files) {
  if (htmlRegex.test(file) === true) {
   fs.readFile(
    path.resolve(__dirname, '../', 'dist', file),
    (error, data) => {
     if (error) {
      throw new Error(error)
     }
     let newHtml = Buffer.from(
      data
       .toString()
       .replace(
        '</head>',
        `<link rel="stylesheet" href="${assets.app.css}"></head>`
       )
       .replace(
```

```
        '</body>',
        `<script type="module" src="${
          assets.vendor.js
        }"></script><script type="module" src="${
          assets.app.js
        }"></script></body>`))
    fs.writeFile(
      path.resolve(__dirname, '../', 'dist', file),
      newHtml,
      error => {
        if (error) {
          throw new Error(error)
        }
        console.log(`Script references injected into ./dist/${file}`)})})}})
```

Модернізації index.html, він надходить до каталогу dist, де він буде використовуватися. Остаточний вигляд файлу наведений в додатку А.

3.3.3. Require

Якщо зберігати всі маршрути в головному файлі index.js, то це все одно що поміщати клієнтський JavaScript-код в HTML-сторінку: це тільки смітить додаток і перешкоджає чіткому розподілу обов'язків. Тому необхідно використовувати систему модулів в Node.js, яка дозволяє писати модульний код.

Всередині модуля Node видимість змінних, оголошення яких починаються з ключового слова var (const), обмежена самим модулем, тому не потрібно писати само виконувану анонімну функцію, щоб не дати змінної просочитися в глобальну область видимості, як на стороні клієнта. Замість цього існує об'єкт module. Значення, присвоєному атрибуту module.exports, стане значенням, яке поверне метод require.

Значення module.exports може мати будь-який тип даних: функція, об'єкт, масив, рядок, число або булева величина. Надалі ми отримуємо можливість викликати повернуту функцію.

Жорстка залежність одного модуля від іншого виникає при використанні require (). Це ефективний, простий і поширений підхід. Серед переваг якого: простота, наочна організація модулів, легкість налагодження.

Проте він має певні недоліки. Труднощі для повторного використання модуля (наприклад, якщо ми хочемо використовувати наш модуль повторно, але з іншим примірником БД). Труднощі для модульного тестування (доведеться створювати фіктивний екземпляр БД і якось передавати його модулю)

3.3.4. CommonJs

Пропозиція модуля CommonJS визначає простий API для оголошення модулів на стороні сервера. Формат був запропонований CommonJS - волонтерською робочою групою, яка має на меті розробити, прототипувати та стандартизувати JavaScript API. На сьогоднішній день вони намагалися ратифікувати стандарти як для модулів, так і для пакетів. З точки зору структури модуль CommonJS - це багаторазовий фрагмент JavaScript, який експортує конкретні об'єкти, доступні для будь-якого залежного коду. За звичай немає обгортків функцій навколо таких модулів.

Модулі CommonJS в основному містять дві основні частини: вільну змінну, що називається, exports що містить об'єкти, які модуль хоче зробити доступними для інших модулів, і require функцію, яку модулі можуть використовувати для імпорту експорту інших модулів.

Декларації про імпорт пов'язують експорт модулів як локальних змінних і можуть бути перейменовані, щоб уникнути зіткнень / конфліктів імен.

Декларації про експорт заявляють, що локальне прив'язування модуля зовні видно таким чином, що інші модулі можуть читати експорт, але не можуть змінювати їх.

3.3.5. WebSocket

Веб-сокети - дивовижна технологія, яка все ширше підтримується браузерами. Веб-сокет дозволяє підтримувати постійний, двонапрямний, не надто ресурсомісткий канал зв'язку між клієнтом і сервером по одному TCP-з'єднанню. В результаті як клієнт, так і сервер отримують можливість в реальному часі посилати повідомлення без накладних витрат і затримок, властивих циклу запит-відповідь в протоколі HTTP. До появи веб-сокетів розробники використовували інші - не настільки ефективні - способи домогтися того ж результату. До них відносяться сокети Flash; довгий час опитування, коли браузер відкриває запит серверу, а потім повторно ініціалізує його, коли приходить відповідь або виникає тайм-аут. І періодичність опитування сервера з невеликим інтервалом (наприклад, один раз в секунду). Недолік веб-сокетів полягає в тому, що остаточна специфікація ще не готова, а старі браузери цю технологію ніколи не будуть підтримувати. Ми користуємося веб-сокетами, коли вони доступні, але якщо це не можливо переходить на інші технології.

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було розглянуто основні модулі та процеси, які необхідні для реалізації веб-додатку планування персональних завдань. Розглянуто файли маніфесту та навіщо їх потрібно використовувати. Для покращення розробки додатку було використано збирач файлів Webpack, який також надає можливість створення локального серверу webpack-dev-server. Визначено головні файлі додатку та описано їх взаємодія. При реалізації додатку відзначено його особливості.

ВИСНОВКИ

В результаті виконання дипломної роботи було спроектовано веб-додаток для планування персональних завдань. Даний додаток повність реалізовано з усім необхідним функціоналом , згідно з поставленою задачею.

В результаті розробки були:

- Отримані навики аналізу та проектування.
- Виявлені особливості одно сторінкових додатків.
- Отримані навики з використання сучасних технологій.

Досліджено архітектуру типового клієнт-серверного додатку. Розглянуто інші архітектурні шаблони як, використовують при створенні веб-додатків. Описано основні технології для розробки під веб та їх популярні надбудови, а також їх доречність для використання в даному випадку. Розглянуто популярні та доцільні методи для створення клієнтської та серверної частини додатку. Визначено чому було використано платформу node.js та як вона взаємодіє з іншими мовами та фреймворками. Реалізований додаток це – SPA. SPA – це додаток, який працює в браузері і не перезавантажує сторінку під час роботи. Даний тип додатків призначений для покращення та оптимізації графічного інтерфейсу користувача. А також як він використовую шаблон MVC. Дане рішення дозволяє розділити систему на три окремі частини:

- **Модель** – в даному випадку, це дані які надходять з серверної частини. Будь-який інтерфейс отриманий з серверної частини , буде отриманий із моделі.
- **Представлення** – це користувацький інтерфейс, який користувач має можливість бачити та взаємодіяти. В даному випадку, він буде генеруватися динамічно на основі поточної моделі додатку.
- **Контролер** – знаходиться на рівні бізнес-логіки та представлення, котрий виконує службові функції.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. developer.mozilla.org [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/How_the_Web_works
2. Специфікація html [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org
3. Документація Sass [Електронний ресурс]. – Режим доступу: https://sass-lang.com/documentation

ДОДАТОК А

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0, shrink-to-fit=no">
  <title>taskmen</title>
  <link rel="icon" type="image/png" href="/img/favicon/favicon-48x48.png" sizes="48x48" />
  <link rel="icon" type="image/png" href="/img/favicon/favicon-32x32.png" sizes="32x32" />
  <link rel="icon" type="image/png" href="/img/favicon/favicon-16x16.png" sizes="16x16" />
  <link rel="apple-touch-icon" sizes="180x180" href="/img/icons/apple-touch-icon.png">
  <link rel="mask-icon" href="/img/icons/safari-pinned-tab.svg" color="#3a7df8">
  <link rel="manifest" href="/manifest.json">
    <style>
    body {
      margin: 0;
    }
    #loading-wrapper {
      text-align: center;
      font-family: -apple-system,BlinkMacSystemFont,"Segoe UI",Roboto,Oxygen-Sans,Ubuntu,Cantarell,"Helvetica Neue",sans-serif;
      padding-top: 2rem;
      color: #666;
    }
    #loading-wrapper p { font-weight: 600; }
```

```
    #loading-button { font-size: 1rem; }
  </style>
<link rel="stylesheet" href="/generated/app.be0ebf27e23a8bd5f2d0.css"></head>
<body>
  <div id="app-shell">
    <div id="loading-wrapper">
      <p id="loading">Loading...</p>
    </div>
    <script>
setTimeout(() => {
  const loading = document.getElementById('loading')
  if (loading) {
    const loadingButton = document.getElementById('loading-button')
    loading.innerText = 'Not connection.'
    loadingButton.style.display = 'inline-block'
  }
}, 3500)
    </script>
  </div>
<script type="module" src="/generated/vendor.1.ffeba11e5f793afcb026.mjs.js"> </script>
<script type="module" src="/generated/app.38d258d60ad9ed7940cd.mjs.js"> </body>
</html>
```

ДОДАТОК Б